TELEPATH TACTICS

Telepath Tactics: An Introduction

Telepath Tactics is a turn-based tactical RPG featuring a Fire Emblem style, story-driven single player campaign as well as a 2-to-6 player multiplayer mode inspired in part by Super Smash Brothers.

Telepath Tactics takes a **deterministic approach** to combat mechanics that is unusual in the world of strategy RPGs. Attack damage is 100% predictable, and by default, attacks always hit unless there is an intervening factor (e.g. the attacker having been blinded, or the target having some special defensive status effect).

To keep things unpredictable, however, the game borrows liberally from other series (including Fire Emblem, Disgaea, and Eternal Poison) to provide a wide variety of **available tactics**, and brings some fresh new **environmental manipulation** mechanics to the table to boot. Cut through bushes; push boulders in the way to block off certain routes; build bridges to create new routes across water or lava; build barricades and summon solid state shields to brunt an incoming attack; destroy walls, doors, and bridges; and place down traps and explosive charges to snare an unwary opponent.

Telepath Tactics multiplayer comes with **23 unique character classes**, each with its own strengths, weaknesses, and battlefield roles. The single player campaign, in turn, features unique, named characters based off of these classes, each with their own custom stats and leveling schemes.

You don't have to be content with the selection the game ships with, however. Telepath Tactics features **extensive mod support** that allows you to create custom multiplayer maps, custom tilesets, custom destructible objects, custom items, custom attacks, custom character classes, custom portraits, custom animations, and even whole single player campaigns filled with unique characters, dialog, and cut scenes.

Read on to learn about Telepath Tactics's setting, learn how the game is controlled, and get a detailed breakdown of the game's rules.

Telepath Tactics Setting

Telepath Tactics takes place in an unusual fantasy world. Some people here possess psychic abilities—collectively referred to as "psy"—but magic, as such, does not exist in this universe. Nor do elves, dwarves, or dragons.

...or horses, for that matter. Most of this world's megafauna are insects—giant mantises, giant scorpions, and giant ant-like creatures called "shadow bugs." Giant mantises and scorpions are often used as mounts, and that is precisely what the cavaliers of this world ride.

Our story begins in the Dundar Archipelago, an expanse of thousands of islands stretching south of the world's equator, originating below the continent of Cera Bella. Due to the sheer size of the Dundar Archipelago, its isles encompass a huge variety of climates and cultures. The unenviable task of trying to govern them all falls to the Dundar empire, a republic perhaps most analogous to ancient Rome.

The Dundar tend to give a wide berth to the lissit, a race of physically imposing bipedal reptiles native to the isles. The lissit live in clans beyond the reach of the empire; their strict warrior culture (and great facility with combat) have dissuaded all but the hardiest of imperial governors from attempting to bring them to heel. Left to their own devices, the lissit tend to ignore human society—and except in cases of over-aggressive settlement by humans, conflict between the races is rare. All the same, many humans in the isles look upon the lissit with fear and distrust.

The Dundar Archipelago is rich in a crystalline mineral called vibra, known for vibrating violently and giving off heat when exposed to a spark. Vibra is effective at heating water, and it is therefore highly sought after for the purpose of powering steam engines.

Enter the shadowlings, a subterranean race of creatures native to mainland Cera Bella. Shadowlings can feed upon human suffering; they have been known to stalk humans (or worse) and psychically torment them in order to feed. The shadowlings are experts at mining vibra, and have a thriving industry based upon bribing various leaders around the world to permit them to mine lands outside the Shadowling Republic's borders.

Despite the fact that humans are naturally prey, the murder and enslavement of humans is nominally illegal among the shadowlings—the practice was outlawed decades ago, after the fall of Queen Nelis at the conclusion of the Shadowling Civil War. However, enslaving humans has a long cultural history among the shadowlings, and large segments of shadowling society cling to the old ways (or simply adopt them to rationalize profiting off of unpaid labor). The Shadowling Republic does not have the resources or the political will to rigorously enforce these laws outside the borders of the shadowlands, and especially not in the innumerable, far-flung islands of the Dundar Archipelago. Meanwhile, money diverted to the right officials ensures that the Dundar will turn a blind eye to any irregularities. Elections are *quite* expensive, after all.

Telepath Tactics Controls

Telepath Tactics features simple, intuitive **mouse controls** for controlling the game's camera.

- Click and drag the ground to pan around the battlefield.
    - If edge panning is enabled in the options, you can also move the mouse cursor to the edges of the screen to pan around.
- Shift-click to focus the battle camera on the spot clicked.
- Click a spot on the minimap to zoom the battle camera to the corresponding location on the battlefield.

The remaining **mouse controls** are designed for quick, streamlined control of your characters.

- *Left-click a character on your team* to select it.

    - The game will display blue movement squares representing every space the character can currently move to.
        - Mousing over a square shows how many steps it will take to reach the square.
        - Click on a blue movement square to move the character there.

    - Whenever a character is selected, you should see a varying set of options in the **Actions Menu**. The buttons shown in the Actions Menu are context-dependent; the game will almost never display an action that that character cannot take! Potential actions in the Actions Menu include:
        - Select next character.
        - Set the selected character to "Done." (This character will be grayed out and will not be able to act again this turn.)
        - Undo the last move or "Done" command.
        - Rotate in place. (Left-click for clockwise; right-click for counter-clockwise.)
        - Move this character.
        - Swim.
        - Grab an item sack.
        - Open the character's Inventory.
        - Open or close a door.
        - Use a battlefield object.
        - Grab a flag.
        - Capture a flag.
        - Rally (move all characters on your team simultaneously).
        - End the turn.
        - Attacks and skills. (These always appear in the bottom row of the Actions menu.)

- *Scroll the mouse wheel* to cycle through your characters.

- *Left-click a character on another team* to see every space it can move to on its next turn, shown in orange squares.
- *Right-click any character or destructible object* to see a detailed summary of all its stats, elemental resistances, and a summary of all items it is carrying.
- *Right-click an empty patch of ground* to deselect the current character.
- *Right-click blue movement square* to remove all movement squares from view.

Telepath Tactics also features **keyboard shortcuts** for most common activities.

- **Arrow keys / WASD** – pan around the battlefield.
- **Space Bar** – autoselect the next available character on your team.
- **Shift + Space Bar** – autoselect the previous available character on your team.

- **Esc** – cancel out of the current menu / deselect the current character / advance dialog.
- **Z** – undo the last character move from this turn. (Telepath Tactics features an undo stack; you can undo as many move commands as you like. Just keep tapping Z.)
- **M** – show the current selected character's move tiles.
- **C** – show a detailed summary of the currently selected character's stats.
- **I** – toggle the inventory menu for the currently selected character.
- **G** – grab item sack.
- **U** – use battlefield object.
- **T** – talk to adjacent character.
- **O** – open/close door.
- **1 - 9** – autoselect whatever attack or skill corresponds to the number pressed for the current selected character.

- **Shift + M** – toggle the minimap on and off.
- **Shift + D** – set current character to “Done.”
- **Shift + E** – end your turn.
- **Shift + O** – display the Objectives screen containing this battle's win and loss conditions.
- **Ctrl + Shift + S** – surrender (in single player, this will restart the battle from the beginning).
- **Ctrl + Shift + Q** – quit to the title screen (only available in single player and in all-CPU multiplayer matches).

- **P** – pause the game.
- **F1** – open up the Options menu (adjust volume, character walk speed, etc.)
- **Ctrl + F** – toggle the game between fullscreen and windowed modes.
- **F10** – take a screenshot of the screen.
- **F12** – take a screenshot of the entire battlefield.
- **Ctrl + F12** – take and automatically format a map preview screenshot.

Telepath Tactics Ruleset

A. The Start of a Battle

- In Multiplayer only: team spawn locations and turn order are each randomized.
- All characters that spawn on Turn 0 are added to the battlefield.
- Each character begins with full Health and 35% of his/her Maximum Energy.
- The first player to move begins by taking his or her turn.

B. The Turn

- At the start of any player's turn, the following happens:
    1. All of that player's characters that rested (i.e. neither moved nor attacked) on the previous turn gain 5 Energy; all characters that moved but did not attack on the previous turn gain 1 Energy.
    2. All of that player's characters have their movement counters and counterattack counters reset.
    3. All of that player's characters scheduled to respawn do so; any reinforcements scheduled to arrive do so.
    4. All of that player's characters that are Burning or Poisoned take damage.
    5. All of that player's non-flying characters sitting on an environmental hazard take damage.
    6. In Multiplayer only: random item drops occur. (See **Item Drops** below.)
- The player may take **Actions** with all of his/her remaining characters.

C. Actions

- A character can take one of ten possible Actions on its turn:
    1. *Move (M)*
        - Every character may move a number of spaces equal to its Speed. (So, for instance, a character with a speed of 5 can move a total of 5 spaces per turn.)
        - Movement is to adjacent spaces only: there is no diagonal movement.
        - Characters may not move through other characters.
            - ♦ There is an option called Ally Pass-Through that permits characters of the same team to move through one another. This is turned off by default. Characters may never land on the same space.
        - Characters do not have to finish moving all of their spaces in one go. A character can move part of its maximum movement range, then move the remainder later on in the turn.
            - ♦ For example: Bob, a Spearman with a Speed of 5, can move 5 spaces over the course of the turn. He can move 2 spaces at the start of the turn, then move another 3 spaces later on after other characters have moved and/or taken actions.
        - Flying characters can move over water and lava; non-flying characters cannot, unless they cross using a bridge or a special movement skill.
        - Flying characters can fly over many types of destructible objects; non-flying characters can move over bridges, flags and item sacks, but most other destructible objects are not passable to them.
        - Non-flying characters may have their movement limited by elevation differences.

(See **Elevation Effects** below.)

2. *Swim (M)*
   - If a non-flying character ever begins a turn in water or lava, that character cannot move normally: instead, it must swim.
   - A swimming character spends 2 Energy to move a single space, either through water or lava, or else onto land from water or lava. After swimming that single space, the character's turn ends.
3. *Undo (Z)*
   - The player can undo character movement; he or she can also undo Movement Skills (such as Leap and Shadowport).
   - Undo becomes available once a character has moved or used a Movement Skill.
   - Undo becomes unavailable, however, should any of the following happen:
     - ♦ the character performs a non-Movement Skill;
     - ♦ the character grabs an item or flag;
     - ♦ the character uses an Item; or
     - ♦ the character reveals tiles covered by fog of war (applicable only in battles where Fog of War is turned on).
4. *Rotate (R)*
   - Character facing matters in Telepath Tactics (see **Backstab** and **Sidestab** below); rotate to switch a character's facing in 90-degree increments.
   - Rotate is a free action: it costs no Energy, and does not end the character's turn.
5. *Skills and Attacks (1-9)*
   - Each character has up to eight distinct Skills at its disposal (attacks are considered a type of Skill).
   - In order to use a Skill, the character must have sufficient Energy to pay the Skill's Cost.
   - Using a Skill produces the Skill's effect on all spaces chosen. (Many Skills can self-target.)
   - Every Skill has a property called *afterAtk* that determines what happens after the Skill is used. There are four types:
     - ♦ **EndTurn** – the character's turn ends.
     - ♦ **CanMove** – the character's turn continues, but the character cannot use any more Skills.
     - ♦ **UseOnce** – the character's turn continues, and the character can continue to use other Skills—but not this one.
     - ♦ **Unlimited** – the character's turn continues, and the character can continue to use any Skills, including using this one again.
   - Most combat Skills are of the EndTurn variety; which is to say, a character's turn will usually end after he or she attacks. However, a few Skills (like Bow or Lance) are classified as CanMove, meaning the character can keep moving after the attack.
     - ♦ For example: Helga the Swordsman has 5 speed. She moves 3 spaces and attacks with Sword. Sword is an EndTurn attack. Therefore, after her attack completes,

her turn automatically ends.

    - ♦ Another example: Billy the Bowman has 5 speed. He moves 3 spaces and attacks with Bow. Bow is a CanMove attack. After his attack completes, he can still move 2 more spaces and rotate freely before ending his turn.
  - Unlike attacks, movement skills tend to be Unlimited. This means that the character can continue taking his turn normally, without restrictions.
    - ♦ For example: Amy the Assassin has 9 speed. She moves 5 spaces and uses the Leap skill (which is Unlimited) to jump over a boulder. She can continue moving up to 4 spaces; she can Rotate; she can attack; or she can even use Leap again!
  - For more information about Skills, see the **Skills** section below.

6. *Grab Item Sack*
   - If a character ever moves onto the same space as an Item Sack, the character may grab it and add its items to his or her inventory.
   - This is a free action: it costs no Energy, and does not end the character's turn.
7. *Inventory (I)*
   - This option appears only if the character has items in his/her inventory; this lets the player inspect all items the character is carrying.
   - If the character has not attacked this turn, the player may use or drop items.
   - This is a free action: it costs no Energy, and does not end the character's turn.
8. *Grab Flag*
   - If a character ever moves onto the same space as a Flag, the character may grab it.
   - If an enemy Flag, it will be added to his/her inventory; but if it's his/her team's Flag, it will automatically return to the flag base.
   - This is a free action: it costs no Energy, and does not end the character's turn.
9. *Capture Flag*
   - This option appears only if the character is (1) carrying an enemy flag, (2) is standing on top of his own flag's base, and (3) his flag's base is currently holding its flag.
   - Selecting this scores a point and automatically returns the enemy flag to its own base.
   - This is a free action: it costs no Energy, and does not end the character's turn.
10. *Done*
    - This ends a character's turn; the character becomes grayed out, and cannot be selected again during this player's turn.

D. <u>Character Stats</u>

- ○ Character stats are as follows:

1. *Health*
   - How much damage the character can take before suffering critical injuries.
2. *Energy*
   - How much energy the character has available to Swim and pay the cost of Skills.
3. *Speed*
   - Determines how many spaces a character can move each turn.
4. *Movement type*

- By default, this is either “flying” or “land”—in essence, whether the character flies or not.
- If a non-flying character is in water or lava, however, his/her movement type becomes “swimming” until he/she emerges onto land.

5. *Strength*
   - Determines the power of this character's physical offensive skills.
6. *Psy Power*
   - Determines the power of this character's psy-dependent offensive skills.
7. *Psy Defense*
   - Determines the power of this character's psy-dependent defensive skills (particularly Shields).
8. *Accuracy*
   - A character's base chance to hit with attacks. This can be modified by blindness, as well as by a target's dodge percentage, but Accuracy is 100% for all characters by default.
9. *Dodge*
   - A character's percentage chance to dodge an attack that would otherwise hit.
10. *Counter*
    - The type of counterattack a character can perform.
11. *Counter limit*
    - The maximum number of counterattacks a character can perform before the beginning of its next turn.
12. *Perception*
    - In battles with Fog of War, this determines how many spaces away a character can see. (It does not have any effect in battles without Fog of War.)
13. *Pushable*
    - Whether a character can be moved by other characters; this is “true” for all characters by default.

E. <u>Skills</u>

- Skills each belong to one of 13 **Elements**:

1. *Slash*
   - Slash is a kind of physical element reserved to Attacks.
   - Swords, axes, and knives usually deal Slash damage.
   - Basic Slash attacks do not have an Energy cost.
   - Slash attacks tend to be effective against psy users, as they have an easier time penetrating psychokinetic shields than other attacks.
2. *Pierce*
   - Pierce is a kind of physical element reserved to Attacks.
   - Spears, arrows and crossbow bolts deal Pierce damage.
   - Basic Pierce attacks do not have an Energy cost.
   - Pierce attacks are often ranged, and deal heavy damage against flying enemies.
3. *Crush*
   - Crush is a kind of physical element reserved to Attacks.

- Maces, fists, and wrenches deal Crush damage.
- Basic Crush attacks do not have an Energy cost.
- Characters heavily armored against Slash or Pierce damage are generally susceptible to Crush damage.

4. *Shield*
   - Shield is an element with both a mental and a physical component.
   - Shield attacks generally either restore character health or buff a character. Shield skills are not offensive in nature.
   - Shield skills are the forte of the Psy Healer class.
5. *Mental*
   - Mental is the only element that is entirely non-physical in nature.
   - Mind Blast, Feedback, Soul Suck and Feint are examples of Mental attacks.
   - Mental skills cannot be dodged or blocked: they *always* hit, even if the attacker is Blinded!
   - Mental attacks are the specialty of the Mentalist and Spirit classes.
6. *Heat*
   - Heat is one of the four psy elements that works via psychokinetic particle manipulation.
   - Heat attacks usually have a base 50% chance to cause Burning status. (See **Status Effects** below.)
7. *Cold*
   - Cold is one of the four psy elements that works via psychokinetic particle manipulation.
   - Cold attacks usually have a base 50% chance to cause Frozen status. (See **Status Effects** below.)
8. *Shadow*
   - Shadow is one of the four psy elements that works via psychokinetic particle manipulation.
   - Shadow attacks usually have a base 50% chance to cause Slowed status, as well as an independent base 50% chance to cause Softened status. (See **Status Effects** below.)
9. *Light*
   - Light is one of the four psy elements that works via psychokinetic particle manipulation.
   - Light attacks usually have a base 50% chance to cause Blinded status. (See **Status Effects** below.)
10. *Create*
    - Create skills spawn a character or object on the battlefield.
    - Create is the primary element of the Engineer class. The Engineer uses it to construct bridges and barricades, as well as to spawn explosive charges.
11. *Movement*
    - Movement skills move the user to the target space, typically bypassing obstacles in the process.

    - Movement skills supplement a character's ordinary maximum movement.
    - The Assassin and Shadowling use Movement skills to reach inaccessible areas.

12. *Falling*
    - This element refers to skills that move other characters around; this includes skills like Shove, Throw, Kinetic Gust and Gravity Spike.
    - Falling attacks deal damage based on falling from heights, and bypass ordinary character defenses.
    - Characters can be knocked, pulled or dropped into environmental hazards using Falling attacks.
13. *Explosive*
    - Explosive attacks are technological in nature (e.g. the detonation of an Engineer's Charge is Explosive).
    - Explosive attacks bypass ordinary character defenses.
    - Explosive attacks deal 5x damage to every destructible object in range (including explosive Charges!) Explosive attacks are far and away the most effective attacks for destroying walls and bridges.

- **Attack damage**
    - Randomized damage does not exist in Telepath Tactics; instead, attack damage is 100% deterministic.
    - Every attack has a base damage calculated from the attacker's Strength, Psy Power, Psy Defense, or some combination of the three. This number is displayed when mousing over an attack's button in the Skills menu.
    - When an attack is launched and it hits a target, the game modifies the base damage in the following order for each character hit:
        1. any elevation bonus or penalty is applied (see **Elevation Effects** > *Effects on ranged attacks* below);
        2. any variable range penalty is applied (see **Ranged Attacks >** *Variable range* below);
        3. any elemental resistance is applied (see **Elemental Resistance** below);
        4. any backstab or sidestab bonus is applied (see **Backstab** and **Sidestab** below);
        5. the final result is rounded to the nearest non-negative integer.
- **Elemental Resistance**
    - Characters of different classes have differing levels of resistance to various elemental attacks, expressed as a percent value.
    - A character's resistance percentage describes how much damage will be subtracted from any attack of that element.
        - ♦ For example: A character with 15% Slash resistance takes 15% less damage from Slash attacks; a character with 50% Heat resistance takes 50% less damage from Heat attacks; and so on.
    - If a character has a *negative* resistance percentage, then that character is *weak* to that element; the same mathematical rule applies!
        - ♦ For example: A character with -30% Cold resistance gets -30% subtracted from

his damage, and thus takes 30% *more* damage from Cold attacks.

- **Backstab**
  - Hitting a character from behind with most attacks will result in bonus damage; this represents the target's inability to adequately defend against the attack.
  - By default, most attacks deal 150% damage during a backstab. Certain Assassin attacks can deal substantially more backstab damage, however.
  - Shields and other non-offensive abilities do not get a backstab bonus.
- **Sidestab**
  - The Assassin's attacks get a bonus to damage when hitting a target in the side. Only the Assassin class gets attacks with a sidestab bonus.
- **Ranged Attacks**

  1. *Elevation effects*
     - (See **Elevation Effects** below.)
  2. *Variable range*
     - Certain ranged attacks can hit at multiple ranges.
     - As a rule, all attacks get a 15% damage penalty for every space beyond the minimum range they are targeted, for a maximum damage penalty of 75%.
       - ♦ <u>For example</u>: a Bowman can normally target 2, 3, or 4 spaces away using the Bow attack, which deals 10 damage by default. 2 spaces is the minimum range for Bow: thus, at 2 spaces, Bow gets no damage penalty, and deals 10 base damage. However if Bow is centered 3 spaces away, the attack gets a 15% penalty, and deals only 8 base damage; at 4 spaces, Bow gets a 30% damage penalty, and deals only 7 base damage. (If Bow can reach 5 spaces away, as with a Bowman positioned in the high ground, it gets a 45% damage penalty there.)
- **Elevation Effects**

  1. *Effects on movement*
     - Non-flying characters cannot move between tiles with an elevation difference greater than 1 except via the use of Skills.
  2. *Falling*
     - Moving from a high elevation to an elevation more than 1 level lower will cause non-flying characters to take falling damage.
     - Likewise, moving a non-flying character onto a wall tile will cause that character to fall until he or she hits the ground.
     - Falling causes 3 damage per point of elevation fallen; falling damage is cumulative with normal attack damage.
       - ♦ For example: a Stone Golem throws Helen from an elevation of 4 to an elevation of 2. Throw deals 4 damage on its own; falling 2 levels of elevation deals an extra 6 damage. Helen takes 10 damage total.
     - Falling 3 or more levels of elevation will cause the character to become Stunned in addition to any damage suffered.
  3. *Effects on melee attacks*
     - A non-flying character may not use attacks with a range of 1 across an elevation difference greater than 1.

4. *Effects on ranged attacks*
    - Characters get one extra space of range on all ranged attacks targeting areas with lower elevation.
        - ♦ E.G. Bill the Bowman is on top of a tower with elevation 3, standing in the northeast corner. To the north and east is grassland with elevation 1; to the west and south are more castle tiles at elevation 3. He will get a range bonus if shooting north or east, but not if shooting south or west.
    - Ranged attacks get a damage bonus or damage penalty based on the relative elevation of the attacker and the target. An attacker with the high ground gets a flat 30% damage bonus; an attacker with the low ground gets a flat 30% damage penalty.

- **Counterattacks**
    - Certain character classes are able to counterattack.
    - Counterattacks are range-dependent: a counterattack will not be triggered if it cannot reach the attacker's space.
        - ♦ For example: Beth the Swordsman has a counterattack that can reach one space. She is hit with a Bow attack from three spaces away. She cannot reach the attacker, and thus will not counterattack.
    - Counterattacks are Energy-dependent: a counterattack will not be triggered if it requires more energy than the would-be counterattacker has.
    - Counterattacks are *not* facing-dependent: a counterattacker *will* turn to face the attacker automatically when counterattacking.
    - Counterattacks are limited to certain number per turn. Once a character has used up all of his/her counterattacks, that character will not counterattack again until his/her counterattack counter is reset. (See **The Turn** above.)
        - ♦ Friendly fire does not trigger a counterattack; characters will never counterattack a teammate.

- **Status Effects**
    - Most status-based effects have a base 50% chance to attach to a character hit with a status-effect-inducing attack. However, any resistance that the target has to the attack's element subtracts from this base chance arithmetically.
        - ♦ <u>For example</u>: Pyro Blast, a Heat attack with a base 50% chance to cause Burning status, hits Sarah. Sarah is a pyrokineticist; as a pyrokineticist, Sarah has 25% Heat resistance. Her Heat resistance (25) is subtracted from the base chance to cause burning (50), leaving a 25% chance that Sarah will be set on fire.
        - ♦ <u>Another example</u>: Pyro Blast, a Heat attack with a base 50% chance to cause Burning status, hits Plato. Plato is a red spriggat; as a red spriggat, Plato has 50% Heat resistance. His Heat resistance (50) is subtracted from the base chance to cause Burning (50), leaving a 0% chance that Plato will be set on fire.
        - ♦ <u>A final example</u>: Cryo Blast, a Cold attack with a base 50% chance to cause Frozen status, hits Plato. Plato is a red spriggat; as a red spriggat, Plato has -25% Cold resistance. The Cold resistance (-25) is subtracted from the base chance to cause Frozen (50), leaving a 75% chance that Plato will be frozen solid.
    - Unlike other status effects, Heavy status attaches 100% of the time as long as the

target is flying.

- Most attacks inflict only one status effect, but the engine supports attacks that inflict numerous different status effects at once.
- The types of Status Effects:
    1. *Weakened*
        - Weakened characters lose 4 physical Strength for 9-12 turns.
    2. *Strengthened*
        - Strengthened characters gain 2 physical Strength for 5-7 turns.
    3. *Lucid*
        - Lucid characters gain 2 Psy Power and 2 Psy Defense for 5-7 turns.
    4. *Stunned*
        - Stunned characters cannot move or act for 1 turn.
    5. *Burning*
        - Burning characters take 4 Heat damage at the start of each turn. Lasts 4-5 turns.
        - Causing a Burning character to become Frozen or pushing him into water will remove Burning status.
    6. *Frozen*
        - Frozen characters cannot move, act, or dodge. Lasts 2-4 turns.
        - Causing a Frozen character to become Burning or pushing him into lava will remove Frozen status.
    7. *Slowed*
        - Slowed characters lose 2 Speed and cannot dodge. Lasts 9-12 turns.
    8. *Hardened*
        - Hardened characters have their resistance to physical damage (Slash, Pierce and Crush) increased by 50. Lasts 5-7 turns.
    9. *Softened*
        - Softened characters have their resistance to physical damage (Slash, Pierce and Crush) reduced by 25. Lasts 5-7 turns.
    10. *Shining*
        - Shining characters have their Dodge increased by 25. Lasts 5-7 turns.
    11. *Blinded*
        - Blinded characters have their attack accuracy reduced by 80, have their Perception reduced to 1, and cannot dodge attacks. Lasts 5-7 turns.
    12. *Enthralled*
        - Enthralled characters come under the control of the current player for the remainder of the turn.
    13. *Levitating*
        - Levitating characters gain the flying movement type for 1 turn.
    14. *Heavy*
        - Heavy characters cannot fly, and will take damage from environmental hazards accordingly.

15. *Poisoned*
    - Poisoned characters take 2 damage per turn for 5-7 turns.
    - Spirits and Golems have no biological circulatory system, and are therefore immune to Poison damage.
16. *Move Bonus*
    - Move Bonus is a positive status effect. A character receiving this status effect gets a bonus to movement for the remainder of the turn equal to 40% of its speed stat rounded to the nearest integer.
17. *Defending*
    - The character has a 50% chance to block all attacks until the start of its next turn.
18. *Back Guarded*
    - Enemy attacks against this character do not receive any backstab or sidestab damage bonuses until the start of this character's next turn.
19. *New Turn*
    - The affected character gets a brand new turn if he or she has already gone; but if this character has not yet gone this turn, nothing happens.
20. *Disarm*
    - The target drops whatever item is equipped to his or her weapon hand.
21. *-1 Counterattack*
    - The target's counterattack limit drops by 1 for one turn. (If it drops to 0, the character cannot counterattack.)

F. **<u>Environmental Hazards</u>**
- Certain tiles, such as water or lava, are hazardous to non-flying characters.
- Any character that begins a turn on an environmental hazard while not flying takes damage from the environmental hazard. (By default, water deals 6 drowning damage per turn and lava deals 20 Heat damage per turn.)
- Characters stranded in an environmental hazard cannot move normally and cannot use Skills: all they can do is access the Inventory, Swim, Rotate, or rest for the turn. Such characters cannot dodge if attacked.

G. **<u>Item Drops</u>**
- Item drops do not occur in Single Player maps.
- In Multiplayer, at the start of every turn, there is a 35% chance that 1-3 item sacks with 1-3 items apiece will spawn at random locations on the battlefield. The further from all player spawn points a space is, the more likely an item drop is to occur there.
- Item sacks cannot spawn on squares with walls or environmental hazards; they *can* spawn on bridges, however.
- Items have a “commonality” property that determines the likelihood they will appear in a drop. A higher commonality value denotes greater likelihood of appearing.
- The commonality value is linear in nature: an item with a commonality rating of 20 is 4 times more likely to appear than an item with a commonality rating of 5.

H. **<u>Experience and Leveling</u>**
- Characters do not gain experience points or level up in Multiplayer maps.

- In Single Player, characters gain:
  - 10 base experience points for using an attack on an enemy;
  - 10 base experience points for moving a character or object;
  - 12 base experience points for creating a character or object;
  - 18 base experience points for healing a hurt ally; and
  - 35 base experience points for slaying an enemy.
- Experience points that you gain from attacking or killing enemies scale depending on your level difference. For each point of level difference, you either gain 10% more experience (if the enemy is higher level), to a maximum of 250% normal experience; or 10% less experience (if the enemy is lower level) to a minimum of 10% normal experience.
  - <u>For example</u>: Alsace is level 4 and has 15 experience points. She attacks a level 6 enemy using Axe, which costs 0 energy. The enemy survives. This gets her 10 experience points for attacking an enemy, which is then scaled up 20% to 12 due to the fact that the enemy is 2 levels higher than her. This puts her at 27 experience points.
- After the base experience is scaled, characters gain an experience bonus based on the energy cost of the skill used; for each point of energy that the skill cost, then character gains an extra unscaled experience point.
  - <u>For example</u>: Emily is level 1 and has 20 experience points. She heals a hurt ally using Mind Shield, which costs 3 energy. This gets her 18 experience points for healing her ally, and 3 points extra for Mind Shield's energy cost, for a total of 21. This puts her at 41 experience points.
- A character that reaches 100 experience points gains a level; the character's experience points are then reduced by 100.
  - <u>For example</u>: John is level 1 and has 80 experience points. He attacks and kills an enemy who is the same level as him using a 0-energy attack. This gets him 10 experience points for the successful attack and 35 points for slaying the enemy, for a total of 45. This puts him at 125 experience points; he levels up and his experience points drop to 25, putting him a quarter of the way to level 3.
  - <u>For example</u>: Emily is now level 1 and has 41 experience points. On her next turn, she attack and kills a level 5 enemy using Mind Blast, which costs 2 energy. This gets her 10 experience points for attacking an enemy and 35 experience points for killing him, both scaled upward 40% due to their level difference. This gets her 63 experience points. She then gets 2 points extra for Mind Blast's energy cost, for a total of 65 experience. This puts her at 106 experience points; she levels up to level 2 and her experience points drop to 6.
- Whenever a character levels up, he or she gains points in exactly two stats. Which stats the character improves are chosen at random, though each character has its own unique set of probabilities of gaining points in different stats.

I. <u>Victory Conditions</u>

- *Last Man Standing*
  - A player wins when no enemies at all remain on the battlefield.
- *Generals*

        - A player wins when no enemy Generals remain on the battlefield.
    - *Capture the Flag*
        - A player wins when he or she reaches the match's point goal through capturing enemy flags.
    - *Defeat Army*
        - Single Player only. The player wins when *one specific* enemy army has no characters remaining on the battlefield.
    - *Defeat the Boss*
        - Single Player only. The player wins when he or she kills a character that the enemy was required to protect.
        - The player may also be required to protect certain characters; the defeat of any of these characters means losing the battle.

J. <u>Deployment</u>

- Sometimes, you will have the option to place your characters on certain spaces at the start of a battle; this is called deployment mode. During deployment mode, you'll have the option to switch which characters you are using, and to switch their starting positions around.
- To deploy a character, simply click its portrait and drag it onto the blue space you want it on. (Note that you can *never* deploy characters to spaces that are not blue!)
- Deployment occurs only on single player maps.

K. <u>Score</u>

- At the end of a battle, the victor's score and battle stats will be displayed onscreen. Player score is calculated as follows:
    1. Base score (150 for a victory, 0 for a loss);
    2. Plus enemies killed times 20;
    3. Minus characters lost times 40 (times 20 if multiplayer);
    4. Plus total points of damage dealt;
    5. Minus total points of damage taken;
    6. Plus items grabbed times 10;
    7. Minus items dropped times 10;
    8. Minus turns taken times 4;
    9. Plus any bonus points / minus bonus penalties.

Classes

There are 23 base character classes in the Telepath Tactics main campaign (though other campaigns may introduce new classes not present in this list):

Swordsman *(promotes to Fencer)*

Lightly armored melee fighters with multiple counterattacks, they are good at going toe-to-toe with single opponents. Swordsmen can sprint great distances if needed, can eliminate enemy counterattacks with feints, and can deal heavy damage with a double strike.

Bandit *(promotes to Marauder)*

Axe-wielders that eschew armor in favor of hard-hitting attacks, bandits can rend enemy armor, hit everyone around them with a whirlwind strike, or gamble on hitting from afar with throwing axes.

Spearman *(promotes to Pikeman)*

Spearmen are heavily armored and can hit from two spaces away with their polearms, or even impale two enemies in a row. In a pinch, they can hide behind their massive shields to try blocking incoming attacks.

Barudit *(promotes to Drake)*

"Barudit" is lissit for "one who fights with a barud," the favored melee weapon of the lissit. Barudit are unarmored, fast, and capable of breaking bones and stunning enemies with their attacks. They can dodge attacks, but are very weak to cold and heat.

Cavalier *(promotes to Mantis Knight)*

Mounted atop giant armored mantises, cavalry are fast-moving, with the ability to strike and continue moving. They can also charge enemies from a distance to knock them back and stun them. Few cavaliers can counterattack, however.

Engineer *(promotes to Machinist)*

Engineers are mediocre fighters but amazing support units; they can build barricades and bridges to access hidden goodies and let you fight the battle on ground of your choosing. They can also lay explosive charges that destroy nearby walls, bridges, and other objects when detonated.

Bowman *(promotes to Bowmaster)*

Unarmored units that can shoot and keep moving, archers are good ranged harassment units. They can also arc their shots or equip long bows for extremely long range attacks that have a chance to miss.

Crossbowman *(promotes to Arbalist)*

Ranged units with a weak melee attack, crossbowmen are hardier than bowmen and can use an explosive bolt to destroy objects on the battlefield and detonate engineers' charges.

Assassin *(promotes to Whisper)*

The fastest unit in the game, assassins can get to just about anywhere they like. Though weak in melee combat, they can leap over obstacles, have a high chance to dodge, and can inflict huge backstab damage upon unarmored enemies.

Shadowling *(promotes to Shadowheart)*

Shadowlings are psychic predators who, in nature, feed on the negative emotions of humans. They fly and can teleport short distances. Shadowlings have natural resistance to mental attacks, and can induce crippling terror in their enemies.

Psy Healer *(promotes to Caduceus)*

The dedicated healer class, psy healers protect wounded units with psychokinetic shields; high-level psy healers can also erect static barriers on the battlefield.

Mentalist *(promotes to Puppetmaster)*

Masters of telekinesis and telepathy, mentalists can push or pull targets at a distance, exercise mind control on enemies, and even make themselves levitate.

Pyrokineticist *(promotes to Pyrokurios)*

A psy with the ability to set enemies on fire with pyrokinetic attacks. At high level, they learn multiple powerful area-of-effect attacks.

Cryokineticist *(promotes to Cryokurios)*

A psy that can freeze enemies with cryokinetic attacks and boost allies' physical resistance. At high level, they learn a powerful area-of-effect attack.

Photokineticist *(promotes to Photokurios)*

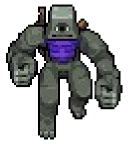
A psy that can blind enemies with photokinetic attacks and give allies a dodge bonus. At high level, they learn a powerful area-of-effect attack.

Red Spriggat *(promotes to Greater Red Spriggat)*

Fast-moving flyers that can breathe fire to hit multiple enemies in a row, or perform hit-and-run attacks with their claws. Strong to heat, very weak to piercing attacks.

Gold Spriggat *(promotes to Greater Gold Spriggat)*

Fast-moving flyers that can breathe a blinding radioactive beam to hit multiple enemies in a row, or perform hit-and-run attacks with their claws. Strong to light, very weak to piercing attacks.



Stone Golem *(promotes to Megalith)*

Towering automatons that resist physical attacks. Stone golems can not only deal heavy damage with their fists, they are also known to grab and hurl enemies great distances. Very slow; weak to light and cold.

Skiakineticist *(promotes to Skiakurios)*

A psy that can slow and soften enemies with skiakinetic attacks, and give allies greater strength. At high level, they learn a powerful area-of-effect attack.

Frost Spriggat *(promotes to Greater Frost Spriggat)*

Fast-moving flyers that can breathe frigid gas to hit multiple enemies in a row, or perform hit-and-run attacks with their claws. Strong to cold, very weak to piercing attacks.

Black Spriggat *(promotes to Greater Black Spriggat)*

Fast-moving flyers that can breathe corrosive, paralytic gas to hit multiple enemies in a row, or perform hit-and-run attacks with their claws. Strong to shadow, very weak to piercing attacks.

Bronze Golem *(promotes to Titan)*

Towering automatons that can withstand a lot of punishment. One arm is a retractable chain with a spinning saw blade capable of ripping through multiple enemies at once. Very slow.

Spirit *(promotes to Specter)*

A humanoid psychic manifestation, spirits can suck health out of enemies and convert it into energy, then transfer their energy to allies as needed. They fly and resist physical attacks, but are quite weak to mental abilities.

Items

There are quite a few items that appear in the main campaign (many of which appear as-is, or somewhat modified, in item drops in multiplayer). Here are a few:

	Name	**Uses**	**Effect**
	Adrenaline Pills	3	+40% to max move for the turn, -3 health
	Apple	1	+10 health
	Bandages	2	+12 health
	Battle Primer, Novice	1	+100 exp. if below level 5; else, +50 exp.
	Battle Primer, Intermediate	2	+100 exp. if below level 12; else, +50 exp.
	Battle Primer, Expert	3	+100 experience points

Caffeine Pills	1	cures slowed status, +10% dodge
Eye Drops	2	cures blinded status, +5 health
Focus Pills	2	+12 energy
Lead Ointment	2	+10% light and shadow res. for the rest of the battle
Splint	3	cures weakened status
Thermal Paste	2	+10% heat and cold res. for the rest of the battle
Whet Stone	1	+1 to strength for the rest of the battle
Wine	2	-20% dodge and accuracy, +20% physical res. for the rest of the battle

Reporting Bugs

If you encounter a bug, do not email me—instead, please report all bugs you encounter to the Telepath Tactics Bugs subforum on SinisterDesign.net! This helps me keep everything nice and organized.

Note that if the game experiences an error, it will automatically generate a *.txt* log file in *Documents > My Games > Telepath Tactics > Logs,* with the filename showing the date and time when the bug occurred. Make sure to include that log file with your report—it'll help me out a lot with fixing the bug! (If you don't see a log file there corresponding to the time the bug occurred, you can force the game to generate a log by hitting the 'L' key.)

The Language of the Lissit

PRONOUNS

si	*see*	I / me
su	*soo*	You
sus	*soose*	We / us
sulesh	*soo - lesh'*	All of us
sek	*sehk*	He / him / it
sekat	*seh - kat'*	She / her
sekesh	*seh' - kesh*	Them / they

TO BE

zetura	*zeh' - tyoo - rah*	To be
si zet	*see zeht*	I am
su zet	*soo zeht*	You are (Imperative: zet = [you] be)
sus zetesh	*zeh' - tehsh*	We are
sek azet	*ah - zeht'*	He is / it is
sekat azet	*ah - zeht'*	She is
sekesh azetesh	*ah' - zeh - tehsh*	They are

TO HAVE

ahsura	*ah' - soo - rah*	To have
si ahs	*ahs*	I have
su ahs	*ahs*	You have (Imperative: ahs = [you] have)
sus ahsesh	*ah - sesh'*	We have
sek ahsas	*ahs - ahs'*	He has
sekat ahsas	*ahs - ahs'*	She has
sekesh ahsasesh	*ahs' - ah - sesh*	They have

POSSESSIVE PRONOUNS

si-dama	*see dah' - mah*	Mine
su-dama	*soo dah' - mah*	Yours
sus-dama	*soose dah' - mah*	Ours
sulesh-dama	*soo - lesh' dah' - mah*	All of ours
sek-dama	*sehk dah' - mah*	His / its
sekat-dama	*seh - kat' dah' - mah*	Hers
sekesh-dama	*seh' - kesh dah' - mah*	Theirs

CONNECTORS

ze	*zee*	The

zesit	*zeh' - sit*	This
so	*soh*	A
iss	*ihs*	And
rosk	*rahsk*	But, however
sor	*sore*	Or
asit	*ah - ssit'*	For
fes	*fess*	Of, about, concerning
isik	*iss' - ihk*	If
risik	*riss' - ihk*	Then

PREFIXES and SUFFIXES

-at	*aht*	(Feminizing suffix)
-esh / -lesh	*ehsh / lehsh*	(Pluralizing suffix; -lesh if word ends in a vowel)
ge'-	*geh*	(Negating prefix, like the English "un-")

RACE NAMES

lissit	*lih'-siht*	A lizardman or lizardman
lissat	*lih - saht'*	A female lizardman
ge'filesh	*geh' - fill - ehsh'*	A shadowling or shadowlings
thirim duuriss	*thih' - rim doo' - riss*	A spriggat
hesh	*hehsh*	A human or humans
silith duuriss	*sih' - lith doo' - riss*	A golem
geduur ambuuriss	*geh - dure' ahm - boohr - ihs'*	A spirit / ghost

GREETINGS

ferat duur	*feh - raht' dure*	Goodbye (*lit.* “remain ferocious”)
su si-ari	*soo see ah' - ree*	Hello (*lit.* “you please me”)

YES and NO

isa	*eeh' - sah*	Yes (also, used interrogatively: “Do you agree?”)
oso	*oh' - soh*	No
esoso	*eh - soh' - soh*	Maybe, Perhaps, Possibly

DIRECTIONS

nuurss	*noohrs*	North
suurss	*soohrs*	South
uurss	*oohrs*	East
wuurss	*woohrs*	West
esat	*eh - saht'*	Right (also: incorrect)
askal	*ahs - kahl'*	Left (also: correct)

MISCELLANEOUS

aduur	*ah' - dure*	Someplace (n.)
aduura	*ah' - dure - ah*	To go (v.); to leave (v.)
adyl	*ah' - dill*	Still, unmoving, passive (adj.)
ahbas	*ah' - bahss*	Bored (adj.)
ahzbulak	*ahz - bool - ahk'*	The Dimly Seen Land; the afterlife of lissit myth (n.)
ahzbu	*ahz' - boo*	Difficult to perceive (adj.)
ahzerik	*ah' - zehr - ihk*	Angry (adj.)
alesh	*ah' - lehsh*	All (adj.)
ambuuriss	*ahm - boohr - ihs'*	Bright, shining (adj.)
ambuur	*ahm - boohr'*	Golden (adj.), gold coin (n.)
ambuura	*ahm' - byoo - rah*	To gild (v.); to beautify, as with gold (v.); to press coins (v.)
ames	*ah' - mayss*	Likewise (adj.); in return, in exchange (adj.)
arima	*ah' - ri - mah*	To please; to make happy (v.)
arimat	*ah - ri - maht'*	Happily, willingly (adv.)
arisit	*ah' - ri - sit*	Lover (n., male)
arisat	*ah' - ri - sat*	Lover (n., female)
barud	*bah - roohd'*	Large, two-handed mace favored by lizardman warriors (n.)
barudit	*bah - roohd - eet'*	One who wields a mace in battle (n.)
barudura	*bah - roohd' - yoo - rah*	To beat or bludgeon (v.)
blud	*bloohd'*	Blood (n.)
bludura	*bloohd' - yoo - rah*	To bleed (v.)
damat	*dah' - maht*	Promise (n.)
damsara	*dahm' - sah - rah*	To take (v.)
damura	*dah' - myoo - rah*	To be obligated (v.)
desara	*deh - sah' - rah*	To obey (v.)
desariss	*deh - sah' - riss*	Obedient (adj.)
dom	*dahm*	Home (n.)
duur	*doohr*	Long-lived, enduring, unending (adj.)
duura	*dooh' - rah*	To live, to be in effect (v.)
duurat	*dooh - raht'*	Life (n.)
duuriss	*dooh' - rihs*	Alive (adj.)
fangiss	*fan' - giss*	Tooth, fang (n.)
fangura	*fan' - gyoo - rah*	To bite; to eat (v.)
ferat	*feh - raht'*	Fierce, ferocious, aggressive (adj.); hot (adj.)
fil	*fihl*	A bite or morsel of food (n.)

filara	*fih - lah - rah'*	To prepare food; to cook (v.)
filesh	*fih - lehsh'*	Meat (n.)
firin	*fih - reen'*	Zealous, passionate (adj.)
gas	*gahss*	Roughly: “the ancestors” (n.)
gasul	*gah - sool'*	Seer (n.)
geduur	*geh - dure'*	Corpse (n.)
geduura	*geh' - dooh - rah*	To kill (v.)
geduuriss	*geh' - dooh - rihs*	Dead (adj.)
gefangura	*geh - fan' - gyoo - rah*	To choke; to fail to eat (v.)
gehsura	*geh' - soo - rah*	To lack; to miss (v.)
gekosh	*geh' - kosh*	Silence (n.)
gekoshiss	*geh - kosh' - ihs*	Silent, quiet (adj.)
gelaan	*geh - lahn'*	Small (adj.)
gestat	*geh - staht'*	Thief (n.)
getor	*geh - tore'*	Weak (adj.)
hass	*hahss*	Honor (n.)
hassiss	*hahss' - ihs*	Graced with honor (adj.)
hass'kalaw	*hahss' - kahl - ow*	Lit. "honor hand"; to give someone hass'kalaw is to bring them honor. (n.)
ho	*hoh*	Day (n.)
hesh	*hehsh*	Hair or fur (n.)
heshiss	*hehsh' - ihs*	Hairy; covered in fur (adj.)
hss	*hiss*	In line for succession (also used as an honorific in place of "resh" for the successor) (adj.)
hsun	*soon*	Protector (n.)
hsuniss	*soon' - ihs*	Protective (adj.)
hsunura	*soon' - yoo - rah*	To protect; to act as protector (v.)
hsunuur	*soon' - ure*	Armor (n.)
idgas	*id - gahss'*	An expression of indifference; it translates roughly to “maybe it matters to the skeletons”
ka	*kah*	Of the lizardmen, in the sense of all lizardmen as a united people (adj.)
kalaw	*kahl - ow'*	Claw, hand (n.)
karok	*kah - rahk'*	Hunger (n.)
karokiss	*kah - rahk' - ihs*	Hungry (adj.)
kosh	*kosh*	Noise (n.)
koshiss	*kosh' - ihs*	Noisy, Loud (adj.)
ko	*koh*	Afraid (adj.); cold (adj.)
kot	*koat*	Coward (n.)
kri	*kree*	Night (n.); darkness (n.)
kta	*ktah*	Fast (adj.); quickly (adv.)
laan	*lahn*	Big (adj.)

lak	*lahk*	“The lands of”; region (n.)
omisigah	*oh - mih - sih - gah'*	a curse word that translates more or less literally to “my ancestors have cursed me”
predat	*preh - daht'*	Hunter (n.)
predura	*preh - dyoo' - rah*	To hunt, to stalk (v.)
ra	*rah*	War, battle, conflict (n.)
ra'kalaw	*rah' - kahl - ow*	Lit. "war hand"; an act of war. (n.)
ra'pet	*rah' - peht*	Warrior, fighter (n.)
ra'pis	*rah' - peese*	Shield (n.)
rasura	*rah' - syoo - rah*	To fight (v.)
resh	*rehsh*	"He who is named" -- used to indicate that a proper name is to follow, like "mister" (n.)
reshat	*reh - shaht'*	"She who is named" -- used to indicate that a proper name is to follow, like "miss" (n.)
sal	*sahl*	Cunning, clever (adj.)
sasara	*sah' - sah - rah*	To give (v.)
scarl	*scahrl*	Red, scarlet (n., adj.), bloody (adj.)
scarlura	*scahrl' - yoo - rah*	To wound; to cause to bleed (v.)
sera	*seh - rah'*	Blade (n.)
serat	*seh - raht'*	Sharp (adj.)
seratura	*seh - raht' - yoo - rah*	To sharpen (v.)
silesh	*sih' - lehsh*	Many (n., adj.)
silit	*sih' - liht*	A crevice (n.); hidden (adj.)
silith	*sih' - lihth*	A boulder (n.); a stone edifice (n.)
silithis	*sih - lihth' - ihs*	Patient (adj.)
simura	*sih - myoo - rah'*	To be able or capable (v.)
so	*soh*	One (n., adj.)
sugeduura	*sooh - geh' - dooh - rah'*	To die (v.)
susugeduura	*sooh - sooh - geh' - dooh - rah'*	To commit suicide (v.)
tal	*tahl*	High, towering (adj.)
t'ayil	*tai - yeel'*	Tail (n.)
thirim	*thih' - rihm*	Temperature (n.)
thuur	*thure*	Danger (n.)
thuuriss	*thure' - ihs*	In danger (adj.)
tor	*tore*	Strong (adj.)
zaris	*zah' - rihs*	Clan (n.)
zerik	*zeh' - rihk*	Anger (n.)
zerikiss	*zeh' - rihk - ihs*	Angry (adj.)
zeruka	*zeh' - ryoo - kah*	To provoke (v.)

zisura	*zih' - syoo - rah*	To come (v.); to approach (v.)

GRAMMAR RULES

Direct Objects

If a verb takes a pronoun as a direct object, the direct object should be attached to the front of the verb with a hyphen.

- For example: "to fight me" would be spoken as "si-rasura"; "to approach him" would be "sek-zisura"; “follow me” would be “si-desar.”

Possessive Nouns

Connect pronouns and nouns with hyphens and the word "dama" in between to denote possession:

- For example: "su-dama-zaris" would mean "your clan"; "Siripent-dama-t'ayil" would mean "Siripent's tail"; and so on.

Past and Future Tense

To denote past or future tense, place “des” or “sid” in front of the verb with a hyphen connector: “des” for past tense, and “sid” for future tense.

- For example: "su sid-geduur" would mean "you will die"; "si des-zet ko" would mean "I was afraid."

The tense designator goes before any pronoun direct object that may also be attached to the verb.

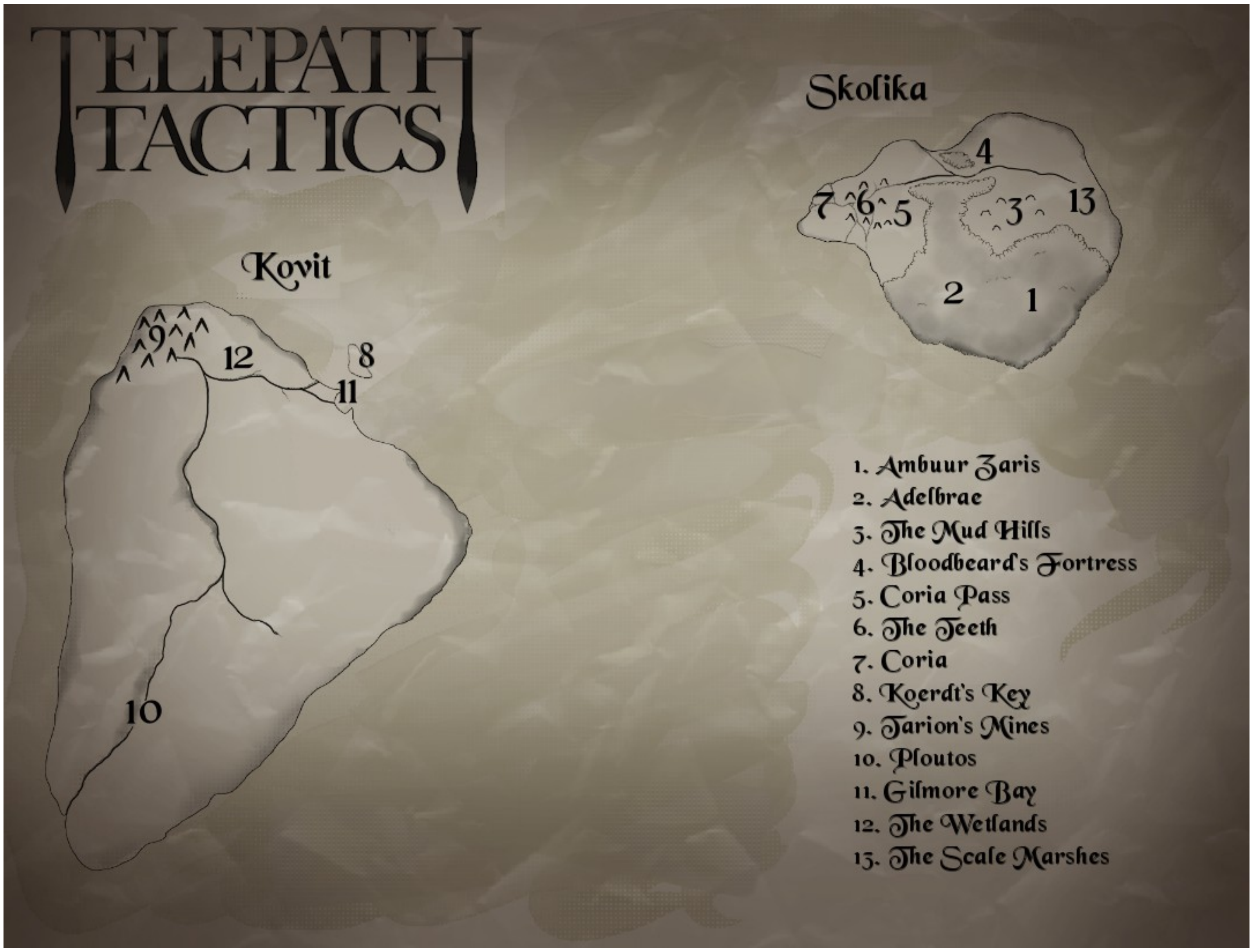
- For example: "sekat sid-sek-ageduur" would mean "she will kill him"; "sekesh des-si-azisesh" would be "they approached me"; and so on. (But note that "they approached my home" would be "sekesh des-azisesh si-dama-doma"--the direct object "my home" is not attached to the verb, since it is not a pronoun.)

USAGE EXAMPLES

- “Alesh ze lissit.”
  1. *All of the lizardmen.*
- “Su gehsesh filesh.”
  1. *We are short on food.*

- "Si zet ahbas fes su, iss kri zisas kta."
    1. *I am bored with you, and night is approaching quickly.*
- "Sekesh des-afangesh Ambuur-dama-filesh!"
    1. *They ate Ambuur's meat!*
- "Sek azet so ra'kalaw!"
    1. *It is an act of war!*
- "Isik ze hesh zerukesh sus, risik sus domesh rasura ames."
    1. *If the humans provoke us, then we must fight back.*

<u>World Map</u>



Telepath Tactics Modding Guide

Telepath Tactics offers extensive support for modding. This section of the manual will explain how to mod various aspects of the game.

Where to find the things

There are two primary places where you can find files to modify: in campaign folders, and in multiplayer rules folders.

- The **campaigns** that come with the game can be found in the game's install directory, in *Data > Campaigns > The Vengeance of Emma Strider, Data > Campaigns > Guard Llama*, and so on.
- User-created campaigns should be found in folders contained in *Documents > My Games > Telepath Tactics > User Campaigns*. If you download a player-created campaign, stick the folder that containing the campaign in this directory. If you create a campaign, create it in its own folder here.

- The **multiplayer rule sets** that come with the game can be found in the game's install directory, in *Data > Multiplayer > Default*, etc.
- User-created multiplayer rule sets should be found in folders contained in *Documents > My Games > Telepath Tactics > User Multiplayer Rule Sets.* If you download a player-created multiplayer ruleset, stick the folder that contains it in this directory. If you create a rule set, create it in its own folder here.

Both campaigns and multiplayer rule sets contain the following basic XML files that you can modify:

1. *AOEPatterns.xml*
2. *Attacks.xml*
3. *CharClasses.xml*
4. *CharNames.xml*
5. *ItemClasses.xml*
6. *ObjClasses.xml*
7. *PersistentDialog.xml*

For a campaign or multiplayer rule set to work, its folder *must* contain all seven of these files!

In addition to these files, each campaign contains levels, contained within a *Maps* subfolder. Multiplayer mode has maps as well, although the multiplayer *Maps* folder exists independently of the individual rulesets.

How to modify the things

First, find a folder for an existing campaign (e.g. The Vengeance of Emma Strider) or multiplayer ruleset (e.g. Default) that you want to use as a starting point. Next, copy the folder. Paste the copy in

Documents > *My Games* > *Telepath Tactics* > *User Campaigns* (if it's a campaign) or *Documents* > *My Games* > *Telepath Tactics* > *User Multiplayer Rule Sets* (if it's a ruleset). Rename the copied folder to whatever you want your campaign or rule set to be called.

Next, start editing the copied files! We'll go through all of the things you can modify now, and where you go to modify them.

A. Attacks

- To mod attacks, open up *Attacks.xml* in a text editor of your choice.
- Note: do not remove the Swim attack from *Attacks.xml*, or the game will not work properly!
- Each attack has the following properties, in order:
  1. **elem** – the attack's element. The game uses this to decide which elemental resistance applies when the attack is launched. Make sure to capitalize the element's name! (Refer to the names of the elements in Section E of the Official Ruleset for a reference.)
  2. **name** – the attack's name.
  3. **d** – leave this alone; it's a placeholder attribute that the game uses when calculating base attack damage. Changing it won't do anything.
  4. **cst** – the attack's Energy cost. If you don't want an Energy cost, set this to 0.
  5. **minRng** – the attack's minimum range. 0 is self-targeting; 1 is one space away; and so on.
  6. **maxRng** – the attack's maximum range.
     - Note: if the attack's element is Move, the range actually defines which spaces the attacker can move onto. Set accordingly!
  7. **shkMag** – this tells the game how strongly to shake the screen when an attack lands. It refers to the maximum number of pixels to move the screen around on each frame; I wouldn't recommend setting this higher than 5. If the attack does not shake the screen, leave this at 0.
  8. **shkTim** – how many frames to shake the screen for on impact.
  9. **strD** – the first of three attributes for calculating base attack power. This is a multiplier based on the attacker's strength. If strength is not used to determine the attack's damage, set this to 0.
     - For example: If this is set to 1 and the attacker's strength is 6, the game will add 6 to the attack's base damage; if this is set to 1.5 and the attacker's strength is 6, the game will add 9 to the attack's base damage; if this is set to 2 and the attacker's strength is 6, the game will add 12 to the attack's base damage.
  10. **powD** – the second of three attributes for calculating base attack power. This is a multiplier based on the attacker's psy power. If psy power is not used to determine the attack's damage, set this to 0.
      - For example: If this is set to 0.5 and the attacker's psy power is 6, the game will add 3 to the attack's base damage; if this is set to 2.5 and the attacker's psy power is 6, the game will add 15 to the attack's base damage.
  11. **defD** – the third of three attributes for calculating base attack power. This is a multiplier based on the attacker's psy defense. If psy defense is not used to determine the attack's damage, set this to 0. If the attack is meant to heal, set its element to Shield and

continue to use positive numbers.

- For example: Jill is a Psy Healer with a psy defense of 12. She uses a Shield attack with a defD of 1.5. The attack will heal 18 damage on the target.

12. **backstabFactor** – the damage multiplier for a striking a character from behind. If this is set to 1 or less, there is no backstab bonus.
13. **sidestabFactor** – the damage multiplier for striking a character in the side. If this is set to 1 or less, there is no sidestab bonus.
14. **selfHealFactor** – the amount the attacker heals upon successful use of the attack, expressed as a multiple of the damage dealt. If the character does not gain health, leave this at 0.
15. **selfFocusFactor** – the amount of Energy the attacker regains upon successful use of the attack, expressed as a multiple of the damage dealt. If the character does not gain Energy, leave this at 0.
    - Note: if selfFocusFactor is set below 0, the game will automatically include it as part of the attack's up-front cost. For example: an attack deals 6 damage, has a cost of 0, and has a selfFocusFactor of -1. Based on the damage, the game will treat the attack's cost as 6 (i.e. 6 damage times the -1 selfFocusFactor).
16. **accMod** – a modifier to the attacker's base accuracy applied with this particular attack. If you want the attack to be less accurate than normal, use a negative number; if you want it to be more accurate, use a positive number. If you do not want the attack to impact the character's chance to hit, leave this at 0.
    - For example: Bella has an accuracy of 100. If she uses an attack with an *accMod* of -20, she will have her chance to hit reduced to 80. If James, with an accuracy of 90, uses the same attack, his chance to hit will be reduced to 70.
17. **statFX** – the effect that you want the attack to confer. (Refer to the names of the status effects in Section E of the Official Ruleset for a reference.) If you do not want the attack to confer a status effect, set this to None.
    - If this is a Create attack, this attribute has a different job! Instead of telling the game what status effect to confer, it instead tells the game whether an object being created has triggers on it.
    - If a Create attack is creating a character, or a destructible object with no triggers, leave this blank. If it's creating a destructible object with a trigger, however, make sure to put two strings separated by a comma: the type of trigger it has, and the name of the script it runs if triggered. The types of triggers follow:
        - ♦ *Pressure* – object is activated as soon as any land-based character steps onto it.
        - ♦ *Switch* – object must be activated deliberately.
18. **affects** – this tells the game what the attack affects. For most attacks, this will be Health. If you want an attack to affect a character's Energy instead, set this to Energy. You can also buff other stats. An attack can affect any of the following:
    - *Health*
    - *Energy*
    - *Max Health*
    - *Max Energy*

- *Speed*
- *Accuracy*
- *Dodge*
- *Strength*
- *Perception*
- *Psy Power*
- *Psy Defense*
- *Slash Res.*
- *Pierce Res.*
- *Crush Res.*
- *Mental Res.*
- *Heat Res.*
- *Cold Res.*
- *Shadow Res.*
- *Light Res.*
- *Poison Res.*

19. **afterAtk** – this tells the game what to do after the attack is used. If the turn ends, use *EndTurn*; if the character can keep moving but cannot attack again, use *CanMove*; if the character can keep moving and use other attacks, use *UseOnce*; if the character can keep moving and use any attacks (including continuing to use this one), use *Unlimited*.
20. **AOE** – this tells the game what area-of-effect the attack has.
    - Use one of the following (unless you've created your own in *AOEPatterns.xml*):
      - ♦ *single* – the attack hits only the targeted space. (Used for most attacks.)
      - ♦ *column_2* – the attack hits two spaces in a row, extending backward from the targeted space toward the attacker.
      - ♦ *column_3* – the attack hits three spaces in a row, extending backward from the targeted space toward the attacker.
      - ♦ *column_3b* – the attack hits three spaces in a row, extending from the attacker forward. (Used for Shield attacks.)
      - ♦ *3x3* – the attack hits nine spaces in a three-by-three square.
      - ♦ *omnidirectional* – the same as 3x3, but with a hollow center. (If minRng and maxRng are both 0, this means the attacker hits every space around him.)
      - ♦ *+* – hits five spaces in a cross configuration.
      - ♦ *omni+* – like omnidirectional above, but with an extra space extending out from the center of each side.
      - ♦ *titan* – like 3x3, but with an extra row of 3 spaces on each side.
      - ♦ *arc_270-0* – hits three spaces in a 90-degree arc from the character's left side to the front.
      - ♦ *arc_90-0* – hits three spaces in a 90-degree arc from the character's right side to the front.
      - ♦ *arc_270-90* – hits five spaces in a 180-degree arc from the character's left side to

    the right side.
    - ♦ *row_split* – hits the two spaces on either side of the targeted space.
    - ♦ *row_3* – hits the targeted space, plus the two spaces on either side of the targeted space.
  - Note: in a Create attack, using any AOE setting other than *single* will create a copy of the thing being created on every single space in the AOE spread!

21. **particles** – this tells the game what particles to spray when the attack lands. Use one of the following:
    - *Sparks*
    - *Blood*
    - *Poison*
    - *Stone*
    - *Wood*
    - *Water*
    - *Smoke*
    - *Spray*
    - *Heat* (will result in both Sparks and Smoke)
22. **targeting** – this tells the game whether the attack is constrained to the four cardinal directions, or whether it can be targeted freely. It is recommended that most attacks leave this set to *constrained*. To permit free targeting, set to *free*.
23. **moveType** – if this attack moves a character, this tells the game what mode the character moves in.
    - There are four moveType settings:
      - ♦ *Normal* – the character moves in a straight line along the ground.
      - ♦ *Parabolic* – the character moves in an arc through the air.
      - ♦ *Teleport* – the character moves in a straight line; this tells the game to ignore walls that would otherwise block the character's movement.
      - ♦ *ToTarget:0* – the character moves adjacent to her target during the attack. If the character cannot do so, the attack is canceled. The "0" represents the number of frames to delay the character's movement by (in case the character does not immediately start moving at the beginning of its attack animation). This setting should *only* be used with constrained, non-Move attacks!
    - If this is a Create attack, however, this attribute has a different job! Instead of telling the game how to move a character, it instead tells the game whether it is placing a character, a normal destructible object, or a bridge. Those three settings are:
      - ♦ *CharPlacement*
      - ♦ *ObjPlacement*
      - ♦ *BridgePlacement*
24. **knockback** – this tells the game how many spaces away to push the target when the attack lands. (A negative number tells the game to pull the character closer.) If you do not want knockback, leave this set to 0.
25. **creates** – if this is a Create attack, place the *charname* or *classname* (if a character

with a charname of *random*) of the character being created here. If creating a destructible object in this way, use the *spritetype* instead. (For instance: if you want to create a Wooden Barricade, put *BarricadeWood*.)

- If this is not a Create attack, you will most likely want to leave this set to *none*. If you put something else in here for a non-Create attack, it will signal to the game that it should spawn the named character or object in place of any character killed by the attack. (For instance: if you have an attack called Petrify, you can set the creates property to *Boulder* to make the game replace any characters killed by Petrify with a boulder.)
- For non-Create attacks, rather than specifying a particular character or object, you can set *creates* to *-CLONE-* to simply replace the dead character with a new copy of itself.

26. **createdOnTeam** – this tells the game whether thing being created is added to the attacker's team, or whether it is an object without a team. If the former, set this to *MyTeam*; if the latter, set this to *Objects*.
    - If this is a non-Create attack and createdOnTeam is not set to either *MyTeam* or *Objects*, the game will spawn the character or object on the same team as the character that was just killed. If this attack's creates property is set to *none*, just leave this blank.
27. **dependsUpon** – name a second attack; this attack will be unavailable to use if the character does not have the second attack as well. For instance: when creating the attack *Swordstorm*, if dependsUpon is set to *Sword*, no character with Swordstorm will be able to use it unless they also have Sword available to use.
    - This is for use on advanced skills which require the character to be equipped with a particular kind of weapon granting a basic attack. (Split Shot, for instance, dependsUpon Bow, which is itself provided via the grantsAtk attribute of an equippable item. So if the character has no bow equipped, the character therefore cannot use Split Shot.)
    - If dependsUpon is left blank, the attack will be available regardless of what other attacks the character is able to perform.
28. **impactFrame** – an integer that counts the number of frames after the end of all animations where the attack actually lands. This tells the game exactly when to initiate screenshake and pop up the animated health bar, etc.
    - Because most attacks appear to hit their targets a few frames *before* the end of their animations, this is usually a negative number! Count the number of frames before the end the attack hits, then multiply by -3. This will usually give you something close to the right number.
29. **soundAndFX** – this contains information about sound effects and visual effects to play during the attack animation. There are two types of tags that get used here, delimited by commas: *SFX[...]* and *VFX[...]*
    - **SFX** – the *SFX[...]* tag allows you to add a sound effect to the animation. There are two parameters that need to be included within the square brackets, delimited by a colon: the name of the mp3 file to be played, and the frame number on which it is to start playing.

  - ♦ Usage example: **SFX[Swoosh Sword:0]** will play *Swoosh Sword.mp3* from the *Data > Sounds* directory beginning on frame 0 of the attack.
- **VFX** – the *VFX[...]* tag allows you to add a visual effect to the animation. There are three parameters that need to be included within the square brackets, each delimited by a colon: the Name of the Animation to Use, the Rule Governing Where the Animation Appears, and the Frame Number on which it is to start playing. There is also an optional Forward/Back Left/Right Offset parameter, with the Forward/Back and Left/Right offset values delimited by a vertical bar (one of these: |).
- There are five Rules Governing Where the Animation Appears for VFX; they are: **OnSelf**, **OnTargets**, **TowardTargets**, **OnCenter**, and **TowardCenter**. They behave as follows:
  - ♦ Usage example: **VFX[MindBlast:OnSelf:12]** will play the *MindBlast.png* animation from the *Data > Characters > Attacks > _VFX* directory, it will play that animation on top of the attacking character, and it will start playing it on frame 12 of the attack.
  - ♦ Usage example: **VFX[Slash:OnTargets:18]** will play the *Slash.png* animation from the *Data > Characters > Attacks > _VFX* directory, it will play a copy of that animation on every space where the attack hits a target, and it will start playing them all on frame 18 of the attack.
  - ♦ Usage example: **VFX[Crossbow Bolt:TowardTargets:35]** will play the *Crossbow Bolt.png* animation from the *Data > Characters > Attacks > _VFX* directory, it will place the animation(s) on top of the attacking character and move it toward the target(s) over the duration of the animation, and it will start playing and moving the animation(s) on frame 35 of the attack.
  - ♦ Usage example: **VFX[DarkVortex_1:OnCenter:21]** will play the *DarkVortex_1.png* animation from the *Data > Characters > Attacks > _VFX* directory, it will place the animation on top of the space where the attack tile was clicked, and it will start playing it on frame 18 of the attack.
  - ♦ Usage example: **VFX[LightBombMissile:TowardCenter:54:30|0]** will play the *LightBombMissile.png* animation from the *Data > Characters > Attacks > _VFX* directory, it will place the animation 30 pixels in front of the attacking character and move it toward the space where the attack tile was clicked over the duration of the animation, and it will start playing it on frame 54 of the attack.
    - ▫ Note: there are three attack frames for every frame of animation in a Telepath Tactics sprite sheet. Thus, if a VFX animation has only one frame, it will travel to its target over the course of three in-game frames. Similarly, if an animation or sound effect is set to play on attack frame 18, it will start alongside the sixth frame of the attack animation.
    - ▫ Note: the Forward/Back Left/Right Offset uses integers. To offset an animation behind the attacker, use a negative value for the first offset; to offset it in front, use a positive value. To offset an animation to the left of the attacker, use a negative value for the second offset; to offset it to the right, use a positive value.

30. **desc** – a short description of the attack that shows up in-game when mousing over the attack button.

B. <u>Characters and Destructible Objects</u>

- To mod characters and character classes, open up *CharClasses.xml* in a text editor of your choice. To mod destructible objects, open up *ObjClasses.xml* in a text editor of your choice.
- Each character or object is enclosed in a Char or Obj tag, respectively. Each has the following properties, in order:
  1. **charname** – the character's name. First name and last name are delimited by a forward slash (e.g. *John/Smith*), not separated by a space. For characters without a last name, simply leave a forward slash on the end of the first name (e.g. *Thallion/*). If creating a destructible object, you can simply leave the charname as *none*. If you set this to *random*, the game will automatically generate a character name at random using the character's **race** and **sex** attributes.
  2. **spritetype** – the name used in the spritesheets for this character. (Since a spiked barricade uses *BarricadeSpiked.png* for its in-game graphic, its spritetype is *BarricadeSpiked*. Since Emma Strider uses the female Swordsman sprite set in *Data* > *Characters*, her spritetype is *Swordsman_F*; and so on.)
     - <u>Note</u>: if the character's *sex* is "Either," leave off the *_M* / *_F* suffix; the game will add one on its own based on the sex it selects.
  3. **portrait** – the character's default portrait within in *Data* > *Characters* > *_Portraits*, minus the .png file extension.
     - <u>Note</u>: if the character's *sex* is "Either," leave off the *_M* / *_F* suffix; the game will add one on its own based on the sex it selects.
  4. **race** – the character's race; used for random name generation and racial item restrictions.
  5. **sex** – the character's sex; used for random name generation. Can be Male, Female, Either or None. (If set to Either, the game will randomly select either Male or Female.)
  6. **classname** – the character's class; used for class item restrictions. For destructible objects, this is displayed in-game in lieu of the object's name (which is always "none").
  7. **move** – movement type. Can be either *land* or *flying*.
  8. **hurtParticle** – what kind of particles does this character or emit when damaged? This is typically set to *Sparks, Stone* or *Wood*; all the particles you can use are listed under the "particles" attribute under attacks above.
  9. **shadowType** – tells the game which image to load for the character or object's shadow. If shadowType is *Small*, the game will load *ShadowSmall.png* out of the Objects folder; if *Big*, it will load *ShadowBig.png*; and so on.
  10. **shadowY** – a number used to adjust the shadow's positioning beneath a character or object.
  11. **charY** – a number used to adjust the character or object's vertical placement on its tile.
      - A higher number means lower placement.
      - By default, all Telepath Tactics characters use a **charY** value of 16; most non-bridge destructible objects are positioned slightly higher, with a **charY** value of 12; walls and doors get a **charY** of 32, meaning that they are positioned with their bottom edges flush with the bottom of the tile; and bridges are positioned even lower, with a

charY value of 36. Experiment to see what looks best with your custom sprites!

12. **lighting** – custom global lighting values that trump the global lighting Condition for whatever level this character or object spawns in. Takes three parameters delimited by commas: Red, Green and Blue values. Just as with Global Lighting parameters, these are decimal numbers that can be anywhere from 0 to 2.0, with 1.0 being 100% color value (0,0,0 will make the character pitch black; 1.6,1.2,0.7 will make the character glow orange; and so on).
    - Note: Leave this blank if you want the character to be lit normally!
13. **ctr** – counterattack; place the name of an attack that the character or object knows if you want it to be able to counterattack with it.
14. **onDeath** – place the name of an attack that you want the character or object to trigger upon death (e.g. Explode). This can be any attack, not just one that the character ordinarily has available to use!
15. **defaultAtkAnim** – when a character attacks, the game automatically looks for a custom animation with the same name as the attack. If it can't find one, it instead loads the defaultAtkAnim.
    - You can find all available animations in *Data > Characters* and *Data > Characters > Attacks.*
    - Psy attacks tend to reuse the same generic "casting" animation, so you'll want to stick *Cast* in for most psy-heavy characters.
    - Never use an animation that doesn't exist for defaultAtkAnim; it can freeze the game. If you aren't sure, just use *Rest*.
16. **atk1** – defines a starting attack that the character knows naturally, without the need for equipment. Use the name of an attack that exists in *Attacks.xml*. There are eight slots for these; leave unused slots blank.
17. **atk2** – see above.
18. **atk3** – see above.
19. **atk4** – see above.
20. **atk5** – see above.
21. **atk6** – see above.
22. **atk7** – see above.
23. **atk8** – see above.
24. **hp** – maximum health.
25. **en** – maximum energy.
26. **spd** – speed; the maximum number of spaces moveable per turn.
27. **ctrLimit** – counter limit; the maximum number of counterattacks a character can launch before the start of its next turn.
28. **dodge** – percentage chance to dodge non-Mental attacks; out of 100.
29. **str** – physical strength.
30. **per** – perception.
31. **psyP** – psy power; strength of character's offensive psy.
32. **psyD** – psy defense; strength of character's defensive psy.
33. **prcRes** – pierce resistance; out of 100.
34. **slshRes** – slash resistance; out of 100.

35. **crshRes** – crush resistance; out of 100.
36. **mnRes** – mental resistance; out of 100.
37. **htRes** – heat resistance; out of 100.
38. **cdRes** – cold resistance; out of 100.
39. **ltRes** – light resistance; out of 100.
40. **shRes** – shadow resistance; out of 100.
41. **poiRes** – poison resistance; out of 100.
42. **acc** – accuracy; base percentage chance to land the character's attacks (i.e. assuming that the target has 0 dodge, attacks have no accMod, and the attacker is not blinded). Out of 100.
43. **lvl** – the character's level; 0 or higher. Used for level item restrictions.
44. **exp** – the character's experience points. (Use a number between 0 and 100.)
45. **pushable** – whether knockback abilities can move the character or object. Can be set to either *true* or *false*.
46. **tags** – this is a sort of catch-all category for miscellaneous modifiers and effects that don't fit elsewhere (status effect immunities and markers that change the game's AI behaviors are all handled with tags). A tag is just a string, sometimes followed by parameters. (If there are parameters, separate them from the tag name using a comma; if there are multiple parameters, delimit each parameter with colons). Individual tags, in turn, are delimited by a forward slash. Scroll down to the **Using Character Tags** subsection below for a complete listing of individual character tags and their attributes.
    - ♦ Usage example: *tags=**"Passive/TargetValue,1.5/ModDmgForMoveType,+:2:flying"*** will add three tags to the character: one which sets that character's AI routines to passive, one which causes other AI characters to treat that character as a more valuable target, and one which gives the character a small damage bonus against flying characters.

- ○ That's all you need to do to create a destructible object or enemy character. To create a character that the player can use, however, you'll need to do one more thing: add leveling information.
    - **Leveling Info**
    - To add character leveling info, make sure there's an <OnLevelUp></OnLevelUp> tag within the Char tags. Each OnLevelUp tag should have the following properties:
        1. **charname** – this must match the character's name from the Char tag exactly.
        2. **hp** – probability of gaining maximum health on level up; a number from 0 to 20, with a higher number meaning higher probability.
        3. **en** – probability of gaining maximum energy on level up; a number from 0 to 20, with a higher number meaning higher probability.
        4. **ctrLimit** – probability of gaining an extra counterattack on level up; a number from 0 to 20, with a higher number meaning higher probability.
        5. **dodge** – probability of gaining extra dodge on level up; a number from 0 to 20, with a higher number meaning higher probability.
        6. **acc** – probability of gaining extra accuracy on level up; a number from 0 to 20, with a higher number meaning higher probability.
        7. **str** – probability of gaining strength on level up; a number from 0 to 20, with

a higher number meaning higher probability.

8. **psyP** – probability of gaining psy power on level up; a number from 0 to 20, with a higher number meaning higher probability.
9. **psyD** – probability of gaining psy defense on level up; a number from 0 to 20, with a higher number meaning higher probability.
10. **prcRes** – probability of gaining pierce resistance on level up; a number from 0 to 20, with a higher number meaning higher probability.
11. **slshRes** – probability of gaining slash resistance on level up; a number from 0 to 20, with a higher number meaning higher probability.
12. **crshRes** – probability of gaining crush resistance on level up; a number from 0 to 20, with a higher number meaning higher probability.
13. **mnRes** – probability of gaining mental resistance on level up; a number from 0 to 20, with a higher number meaning higher probability.
14. **htRes** – probability of gaining heat resistance on level up; a number from 0 to 20, with a higher number meaning higher probability.
15. **cdRes** – probability of gaining cold resistance on level up; a number from 0 to 20, with a higher number meaning higher probability.
16. **ltRes** – probability of gaining light resistance on level up; a number from 0 to 20, with a higher number meaning higher probability.
17. **shRes** – probability of gaining shadow resistance on level up; a number from 0 to 20, with a higher number meaning higher probability.

- To have the character learn new attacks upon reaching certain levels, sticking attack names and level numbers into the space between the OnLevelUp tags, delimited by commas.
  - ♦ Usage example: **<OnLevelUp … >Shove,2,Feint,3</OnLevelUp>** will cause the character to automatically learn the Shove attack upon reaching level 2 and learn the Feint attack upon reaching level 3. (Don't actually use ellipses in your tag; those are just standing in for all the properties above in order to save space.)

○ **Using Character Tags**

- Within the context of XML formatting, a tag refers to something enclosed by brackets (these things: <>). Within the context of character attributes, however, "tags" refers to strings (sometimes with attached parameters) that attach to characters and destructible objects and tell the game to treat them differently. Most tags are either AI-related, or else concern immunity from status effects. To use a tag with parameters, separate the parameters from the tag name with a comma, then delimit the parameters with colons. A full listing of the game's supported tags follows:

1. **Health, Energy, Max Health, Max Energy, Speed, Dodge, Strength, Perception, Psy Power, Psy Defense, Accuracy, Pierce Res., Slash Res., Crush Res., Mental Res., Heat Res., Cold Res., Light Res., Shadow Res.,** *and* **Poison Res.** – each of these directly modifies the corresponding character stat. Takes one parameter: a positive or negative integer to be added to the affected stat.
   - ▫ Note: these tags *only* modify a stat at the moment a character spawns. If a

character receives one of these tags after spawning, it will *not* affect his or her stats until the next battle.

2. **ID** – gives a character a unique identifying number, to be used with the special character *ID[]*. Takes one parameter: the number to give.
3. **IgnoreArmy** – an AI tag with one parameter: a non-negative integer corresponding to an existing army number. This causes the tagged character to ignore the existence of all characters belonging to a particular army; they will be treated as neither enemies nor allies.
4. **LevelUp** – a tag that tells the game to make the tagged character silently gain one or more levels as soon as he or she spawns on the battlefield. This is a quick and easy way to create tougher versions of existing enemy types, as well as ones that provide more experience points. One parameter: a positive integer, Levels To Gain.
   - Usage example: **LevelUp,4** will turn a level 1 enemy into a level 5 version of that enemy.
5. **ModCostForAttack** – a tag that tells the game to modify the energy cost for the tagged character to use the named attack. Takes three parameters: operator type, amount, and attack name.
   - Usage example: **ModCostForAttack,%:75:Cryo Blast** will decrease the cost to use Cryo Blast to 75% of its normal amount for the tagged character.
6. **ModDmgForAttack** – a tag that tells the game to modify the character's attack damage whenever it uses the named attack. Takes three parameters: operator type, amount, and attack name.
   - Note: the modification occurs before resistance is applied in the damage calculation.
   - Usage example: **ModDmgForAttack,+:1:Mind Blast** will increase the character's damage by 1 whenever he/she attacks with Mind Blast.
7. **ModDmgForClass** – a tag that tells the game to modify the character's attack damage based on the Class of the target. Takes three parameters: operator type, amount, and class name.
   - Note: the modification occurs before resistance is applied in the damage calculation.
   - Usage example: **ModDmgForClass,+:5:Cavalier** will increase the character's damage by 5 against cavaliers.
8. **ModDmgForMoveType** – a tag that tells the game to modify the character's attack damage based on the Move type of the target. (This tag only works against characters, not destructible objects.) Takes three parameters: operator type, amount, and move type.
   - Note: the modification occurs before resistance is applied in the damage calculation.
   - Usage example: **ModDmgForMoveType,*:2:flying** will double the character's damage against flying characters.

9. **ModDmgForRace** – a tag that tells the game to modify the character's attack damage based on the Race of the target. Takes three parameters: operator type, amount, and race.
   - Note: the modification occurs before resistance is applied in the damage calculation.
   - Usage example: **ModDmgForRace,-:3:Shadowling** will decrease the character's damage by 3 against characters whose race is Shadowling.
10. **ModDmgForTag** – a tag that tells the game to modify the character's attack damage based on whether the target has a tag of a certain type. Takes three parameters: operator type, amount, and tag name.
    - Note: the modification occurs before resistance is applied in the damage calculation.
    - Usage example: **ModDmgForTag,%:50:Passive** will halve the character's damage against characters with a Passive tag.
11. **ModRngForAttack** – a tag that tells the game to modify the character's maximum range whenever it uses the named attack. Takes three parameters: operator type, amount, and attack name.
    - Usage example: **ModRngForAttack,*:1.4:Bow** will increase the character's maximum range with the Bow attack by 40%.
12. **Passive** – an AI tag; no parameters. This affects the tagged character's AI routines. The character will not move during its turn unless it detects an enemy within its move-and-attack range.
13. **Promoted** – a tag with one parameter: an integer corresponding to the number of levels the character possessed prior to its promotion. This causes experience scaling to work properly with characters who have been promoted and had their level reset to 1.
    - Usage example: **Promoted,19** will treat a character as 19 levels higher than his or her Level stat for purposes of experience scaling (representing the fact that the character was promoted to a new class upon reaching level 20).
14. **RangeBonus** – a tag which increases the range of all ranged attacks (i.e. attacks which otherwise have a maximum range greater than 1). One parameter: the number of spaces to extend the range of ranged attacks.
    - Usage example: **RangeBonus,1** will increase the maximum range of all the tagged character's ranged attacks by 1. For a crossbowman, this would mean that Crossbow and Powder Bolt can hit up to one space further away, but Bayonet cannot.
15. **Selectable** – a tag that is only used in *CharClasses.xml* in multiplayer rulesets. When a character class is given this tag, it lets the game know that that class can be added to players' army rosters. (Classes that lack this tag will not be available to add to the players' army rosters, but may nonetheless be used in maps, or be created during battles via certain attacks.)
16. **TargetValue** – an AI tag with one parameter: a number (including decimals) that acts as a value multiplier. This affects the tagged character's perceived

value in other characters' AI routines. A higher value means that other characters controlled by the AI are more likely to choose this character as a target for skills and attacks; a lower value means the character is more likely to be ignored.
    - Usage example: **TargetValue,2** will double the character's perceived value as a target.
17. **TreatAsArmy** – an AI tag with one parameter: a non-negative integer corresponding to an existing army number. This tag is solely for destructible objects; it causes the AI to treat the object as if it were a character belonging to the designated army.
    - Usage example: **TreatAsArmy,0** will cause the tagged destructible object to be treated as a character within Army 0 (which, by default, means that all non-allied CPU-controlled armies will consider it an attack target).
18. **Trigger** – a tag solely for destructible objects which gives the tagged object a new trigger. This is mainly intended for use in ObjClasses.xml, as a way to consistently tie triggers to certain objects (traps, for example). Takes two parameters: trigger type and script name.
    - Usage example: **Trigger,Pressure:Snare Trap** will cause the tagged destructible object to employ a pressure trigger that runs a script named Snare Trap when activated.

C. **Conditions**

- Conditions set the starting conditions for a map when it is loaded. They can affect anything from global lighting to fog of war, permadeath to victory conditions.
- **Types of Conditions**

1. **Ally Pass-Through** – one parameter, *true* or *false*. (False by default.) If set to *true*, characters from the same army will be able to move through one another's spaces.
    - ♦ Usage example: **<Condition>Ally Pass-Through,true</Condition>** will turn on ally pass-through for this battle.
2. **Defeat Army** – two parameters: the number of the army whose victory condition this affects, and the enemy army that the first army must defeat. Without this condition in place, each army can win a kill-all victory only by wiping out *every* enemy army on the battlefield; with this condition in place, the army specified in parameter 1 need only defeat the one army specified in parameter 2 to win.
3. **Defeat Scene** – one parameter: scene name. Rather than just restarting the battle, make the game proceed to the specified scene upon the player's defeat.
4. **Delay Maneuvers** – two parameters, *army number* and *number of turns*. Causes any AI-controlled player in charge of the designated army to automatically skip each of its turns until it has passed the designated turn number.
    - ♦ Usage example: **<Condition>Delay Maneuvers,1,3</Condition>** will cause the enemy AI controlling army 1 to skip its first three turns.
5. **Deployment** – one parameter, *true* or *false*. (False by default.) If set to *true*, the player will have the opportunity to drag characters onto the battlefield, and swap them around between possible spawn locations.
    - **Note**: There is an optional second parameter, Deployment Music. This tells the game

to play a particular track during deployment; once deployment is over, the game reverts to playing the music designated in the map's *musictrack* attribute.

6. **Exploration Mode** – one parameter, *true* or *false*. (False by default.) If set to *true*, the game makes it so the player's characters can move around continuously without ending the turn. (It turns off features like undo, rotate and end turn, makes it so none of your characters lose steps while moving, restricts access to most attacks, and gives everyone a move range of 12, in addition to automatically turning off Kill-All Victory (see below).
    - ♦ Usage example: **<Condition>Exploration Mode,true</Condition>** will turn on exploration mode in this scene.
7. **Fatigue** – takes two parameters: *army number* and *starting fatigue level*. Army Number is the army whose fatigue you want to set; you can use -1 to have the game set fatigue for all armies on the battlefield. Starting Fatigue Level is a number between 0 and 1 representing the percentage of a character's maximum Energy that will be missing at the start of a battle.
    - Starting energy level is the inverse of starting fatigue: 0 is 0% fatigue, meaning that the characters in the chosen army will start with full Energy; 0.3 is 30% fatigue, meaning that characters in that army will start with 70% Energy; 1 is 100% fatigue, which means 0% starting Energy; and so on.
        - ♦ Usage example: **<Condition>Fatigue,0,0.52</Condition>** will cause army 0's characters to start the battle with 48% Energy.
8. **Fog of War** – one parameter, *true* or *false*. (False by default.) If set to *true*, the battlefield will be covered by a fog of war.
    - ♦ Usage example: **<Condition>Fog of War,true</Condition>** will turn on fog of war for this battle.
9. **Global Lighting** – one parameter, *name of lighting preset*; if "Custom" is used as the lighting preset name, there are three extra parameters: *red, green and blue* with values between 0 and 1. Available global lighting presets include:
    - *Cave*
    - *Daylight*
    - *Dawn*
    - *Evening*
    - *Grayscale*
    - *Indoors* (default)
    - *LavaDark*
    - *LavaLight*
    - *Night*
    - *Overcast*
    - *Sepia*
        - ♦ Usage example: **<Condition>Global Lighting,Night</Condition>** will cause the battlefield to be lit as if it were nighttime.
        - ♦ Usage example: **<Condition>Global Lighting,Custom,0.7,0.7,1</Condition>** will reduce red and green values by 30% across the battlefield, causing

everything to be lit dark blue.

10. **Go First** – one parameter, *army number.* This army goes first in this battle. Single-player only.
11. **Kill-All Victory** – one parameter, *true* or *false*. (True by default.) If set to *false*, the game will not end the battle as soon as it detects that no enemies remain on the battlefield.
12. **New Army** – three parameters, *army name, human or CPU,* and *army color.* Each of these parameters is a string. For *human or CPU,* write Human or CPU; any other value for this parameter will be interpreted as establishing a CPU player. (For available army colors, see **Set Army Color** below.)
    - Note: If this is the first New Army condition in the map, it will apply to army 2; the next New Army condition will create settings for army 3, and so on.
    - Note: You can use an optional fourth parameter to give this army a roster. Use slash notation for character names, and delimit each character with a colon.
        - ♦ Usage example: **<Condition>New Army,The Green Hunters,Human,Green</Condition>** will create a new army called The Green Hunters; the army will be human-controlled, and characters in this army will use a Green team color palette.
        - ♦ Usage example: **<Condition>New Army,Traitors,CPU,Black, General/Vile:Leela/Vile:Emilio/Vile</Condition>** will create a new army called Traitors; the army will be computer-controlled, characters in this army will use a Black team color palette, and the roster will be populated by General Vile, Leela Vile, and Emilio Vile.
13. **Permadeath** – one parameter, *true* or *false*. (True by default.) If set to *false*, characters that reach 0 health will automatically come back in the next scene.
14. **Post-Battle Looting** – one parameter, *true* or *false*. (False by default.) If set to *true*, the game will check for any loose item sacks lying around as soon as the battle ends. If it finds one or more, if the player won and it's a campaign battle, the game will enter exploration mode with a prompt to the player to grab loot and then click the Leave Battlefield button in the Actions Menu.
15. **Protect Char** – two parameters, *army number* and *name of character to protect*. The designated army must protect the named character throughout the battle. If the named character dies, the designated army will lose the battle immediately!
        - ♦ Usage example: **<Condition>Protect Char,0,Captain Duddly</Condition>** will make it so the player loses if Captain Duddly dies.
        - ♦ Usage example: **<Condition>Protect Char,1,General Evil</Condition>** will make it so the enemy loses if General Evil dies.
16. **Roster Number** – one parameter, *roster number* to pull characters from for Army 0 when filling in "FromPlayerRoster" spots on the map. (By default, this is set to roster 0.)
17. **Set Army Alliance** – two parameters, *army number* and *name of alliance*. The designated army becomes a member of the named alliance. To ally two or more armies, just set each of them to the same alliance.
        - ♦ Usage example: **<Condition>Set Army Alliance,0,Imperials</Condition>** and **<Condition>Set Army Alliance,2,Imperials</Condition>** will ally armies

0 and 2 under the alliance name “Imperials.”

18. **Set Army Color** – two parameters, *army number* and *name of color.* The designated army's color is changed to the color in the second parameter. Color names the game will accept include:
    - *Red*
    - *Blue*
    - *Green*
    - *Yellow*
    - *Violet*
    - *Pink*
    - *Orange*
    - *White*
    - *Black*
19. **Set Army Name** – two parameters, *army number* and *name of army.* The designated army's name is changed to the name in the second parameter.
    - ♦ Usage example: **<Condition>Set Army Name,0,Emma's Raiders</Condition>** will rename army 0 to “Emma's Raiders.”
20. **Space Bonus** – two or more parameters: *y position, x position,* and an unlimited number of additional parameters specifying stat bonuses or penalties for characters who stand on that space. Every 2 parameters you add on after the first two specifies an additional bonus or penalty: the third / fifth / seventh etc. will be the name of the stat, and the fourth / sixth / eighth etc. will be the amount of the bonus / penalty. Stat names the game will accept for purposes of bonuses and penalties include:
    - *Perception*
    - *Strength*
    - *Psy Power*
    - *Psy Defense*
    - *Accuracy*
    - *Dodge*
    - *Resistance* – this affects every type of resistance simultaneously.
    - *Slash Res.*
    - *Pierce Res.*
    - *Crush Res.*
    - *Mental Res.*
    - *Heat Res.*
    - *Cold Res.*
    - *Shadow Res.*
    - *Light Res.*
    - *Poison Res.*
        - ♦ Usage example: **<Condition>Space Bonus,4,6,Dodge,10,Pierce Res.,10,Strength,-1</Condition>** will cause the space 4 from the top of the map and 6 right from the left edge to give whatever character stands on it a 1 point

Strength penalty, but +10 to Dodge percentage and +10 to Pierce Resistance percentage.

21. **Victory Scene** – one parameter: scene name. Overwrite *nextbattle* with the specified scene.
22. **Weather** – one parameter, *weather type*. Creates a weather overlay of the chosen type. Available weather types include:
    - *Rain*
    - *Snow*
    - *Fire*
    - *Embers*
    - *Pollen*
        - ♦ Usage example: **<Condition>Weather,Fire</Condition>** will cause fiery sparks to rain down from the sky.
        - ♦ Usage example: **<Condition>Weather,Embers</Condition>** will cause fiery embers to float upwards.

D. Dialog

- ○ Character dialog occurs within maps (though it can also be stored in *PersistentDialog.xml*). To edit dialog, use the map editor and enter "Edit Dialog" mode by pressing the speech bubble icon.
- ○ Dialog in a map is organized into distinct conversations, each with its own dialog trigger.
- ○ Right-click an empty spot in the dialog map to create a new conversation branch.
- ○ To edit a conversation branch, left-click the box that represents it. An edit window will open. In the window, you can:
    1. Change the trigger and trigger parameters.
        - Note: all branches in a conversation *must* share the same trigger and trigger parameters. If you change the trigger or trigger parameters for one branch in a conversation, the editor will stick that branch on the end of a new conversation!
    2. Change the name of the speaker.
    3. Change the body of text dialog.
    4. Add, edit, or remove scripted Actions.
        - Actions are little scripts that can be placed on a conversation branch to make different things happen in the game when the player reaches that branch. Types of Script Actions are listed below.
    5. Add, edit, or remove Replies.
        - Replies determine what happens when the player clicks to continue the conversation. If there is only one reply, it won't be displayed visually in the conversation; if there is more than one reply, all replies will be displayed for the player to select from.
- ○ When you're finished editing, just close the edit window; your changes will be reflected in the dialog map.
- ○ **Special Characters in Dialog Text**
    1. typing **--** will create an em dash (a single, long dash that looks like this: —)
    2. typing **-N-** will skip a couple of lines to start a new paragraph
    3. typing **-NN-** will skip four lines

4. typing **-TURN-** will display the total number of player turns passed, counting each player's turn individually. (This corresponds to the turn parameter in an OnTurn trigger.)
5. typing **-FULLTURN-** will display the total number of times that every player on the battlefield has gone.
6. typing **-Y-** will cause the game to substitute the Y coordinate of the character who triggered the dialog (for dialog which is character-triggered).
7. typing **-X-** will cause the game to substitute the X coordinate of the character who triggered the dialog (for dialog which is character-triggered).
8. typing **-NAME-** will substitute the first name of the character who triggered the dialog (for dialog which is character-triggered).
9. typing **-LNAME-** will substitute the last name of the character who triggered the dialog (for dialog which is character-triggered).
10. typing **-FNAME-** will substitute the full name of the character who triggered the dialog (for dialog which is character-triggered).
    - Note: **-FNAME-** can be nested within **-VAL-**, **-STR-** and **-STAT-**.
11. typing **-ATTACKER-** will substitute the full name of the character who is attacking as of the moment the dialog is triggered (for dialog which is triggered during an attack).
12. typing **BNAME[*x*]** in a script action will substitute the branch number for whatever conversation branch is named *x*. This will let you navigate to branches by name using the game's "GoTo" script actions (see "Types of Script Actions" below).
    - Usage example: **GoTo BNAME[Rejected Offer]** will look for a conversation branch named *Rejected Offer*. If the game finds it, it will use that branch's number in lieu of the special character, then go to that branch number; if no branch has that name, however, the game will use -1 instead and end the conversation.
13. typing **ID[*x*]** in a script action will tell the game to look for the character with an ID tag of the number *x*. This can be used in any script action parameter where you'd ordinarily use a character's name, but will not work in any other context!
    - Usage example: **MoveChar ID[12],6,6** will move whichever character has been tagged with ID 12 to the coordinates 6 , 6.
14. typing **ITEMVAL[*x*]** will substitute the default monetary value of whatever item is named *x*.
15. typing **R[*x-y*]** in a script action will tell the game to randomly select a whole number between *x* and *y*. Both *x* and *y* must be non-negative integers.
    - Usage example: **You rolled a R[1-6].** will select a whole number between 1 and 6 and display it in place of "R[1-6]".
16. typing **-VAL:Blah-** will cause the game to replace it with the value of the custom variable named "Blah"; obviously, you'll be typing the name of an actual custom variable there, not Blah (unless you created a variable named Blah for some reason). If you haven't set a variable by that name yet, the game will just display a 0 by default.
    - This special character can be used in Replies and Actions as well.
    - Usage example: **You have -VAL:Money- gold left.** will read as "You have X gold left," where X is the value of the custom variable Money. If Money is currently 55, for instance, the dialog will read "You have 55 gold left."
17. typing **-STR:Blah-** will cause the game to replace it with the string stored under the

"Blah" custom variable name.

  - This special character can be used in Replies and Actions as well.
  - Usage example: **We are allied with -STR:Ally-.** will read as "We are allied with X," where X is the string stored in the custom variable Ally. If Ally is currently storing "The Resistance", the dialog will read "We are allied with The Resistance."

18. typing **-ARR:Array Name,Get From-** will cause the game to replace it with the specified array entry stored in the named custom array.
19. typing **-STAT:Character Name,Stat-** will replace "Character Name,Stat" with the value of the named stat for the named character.
    - You can use every stat supported by the **IfStatGoTo** action for this; see **IfStatGoTo** below. If the game can't find the named character on the battlefield or you don't correctly type the name of an actual stat, the game will just display a 0 by default.
    - This special character can be used in Replies and Actions as well.
    - Usage example: **Emma has -STAT:Emma Strider,Slash Res.-% resistance to Slash damage** will read as "Emma has X% resistance to Slash damage," where X is Emma Strider's current Slash Resistance. If her Slash Resistance is currently 10, for instance, the dialog will read "Emma has 10% resistance to Slash damage."

- **Types of Triggers**

1. **OnLoaded –** takes no parameters. Causes dialog to trigger the moment the level is fully loaded. Use this with various script actions followed by EndConvImmediately to alter the level before the player can see it happening.
2. **OnTurn** – takes one parameter, an integer. Causes dialog to trigger at the very start of the turn represented by the parameter. (If *-1* is used, it will trigger on any turn.)
    - ♦ Usage example: **OnTurn 0** will cause the conversation to trigger at the start of turn 0 (i.e. at the very beginning of the battle, before the player moves); **OnTurn 1** will cause the conversation to trigger at the start of turn 1 (i.e. before the second player moves for the first time).
3. **OnTalk** – this allows for a player-triggered conversation. It takes two parameters: the names of two characters. When one character is adjacent to the other, the Talk option will appear in the actions box. If the player clicks it, it will trigger the conversation.
    - Note: if you want any character to be able to initiate the conversation, you can use *-ANY-* for the first parameter.
      - ♦ **Usage example: OnTalk Emma Strider,Grant Winter** will cause the game to present the Talk option when these characters are adjacent. If the player clicks Talk, the associated conversation will begin.
4. **OnReachingSpace** – this triggers a conversation as soon as a character reaches a defined space on the battlefield. It takes between two and four parameters: the Y coordinate, the X coordinate, the name of which character must trigger the conversation (if it can only be triggered by a particular character), and the team number of the character who must trigger the conversation (if it can only be triggered by a character of a particular team).
    - Note: if you want to make it so *only* destructible objects can trigger the conversation, set the required team number to 99; if you want to make so *only* characters (not destructible objects) can trigger the conversation, set it to -99.

- ♦ __Usage example:__ **OnReachingSpace 4,6** will trigger the conversation when any character ends a move on the coordinates 4,6.
- ♦ __Usage example__: **OnReachingSpace 4,6,Emma Strider** will trigger the conversation when Emma Strider (and only Emma Strider) ends a move on the coordinates 4,6.
- ♦ __Usage example__: **OnReachingSpace 4,6,,0** will trigger the conversation when any character on team 0 (the player's team) ends a move on the coordinates 4,6.

5. **OnOpeningDoor** – this triggers a conversation as soon as a door at a defined space on the battlefield is opened. It takes two parameters: the Y coordinate of the door, and the X coordinate of the door.
    - ♦ __Usage example:__ **OnOpeningDoor 4,6** will trigger the conversation when a door at the coordinates 4,6 is opened.
6. **OnLockedDoor** – this triggers a conversation as soon as a character attempts to open a locked door at a defined space on the battlefield without having a key. It takes two parameters: the Y coordinate of the door, and the X coordinate of the door.
7. **OnUsing** – this triggers a conversation as soon as a Use trigger at a defined space on the battlefield is tripped. It takes two parameters: the Y coordinate of the trigger, and the X coordinate of the trigger.
    - __Note__: if a conversation is triggered in this way, the script attached to the Use trigger will *not* automatically run; you'll need to run it manually from within the dialog instead if you want it to run.
        - ♦ __Usage example:__ **OnUsing 8,4** will trigger the conversation when a Use or Use Once trigger at the coordinates 8,4 is tripped.
8. **OnGrab** – this triggers a conversation as soon as a character grabs an item sack on the battlefield. It takes up to five parameters: the name of the character that must grab the item sack to trigger the conversation (if it can only be triggered by a particular character), the team number of the character who must trigger the conversation (if it can only be triggered by a character of a particular team), the sack's Y coordinate (if that matters), the sack's X coordinate (if that matters), and the name of an item that the sack must contain to trigger the conversation (if it can only be triggered by a sack containing a particular item).
    - __Note__: *all* of these parameters are optional! To have the game ignore team number or sack Y or X coordinates, use *-1*; to ignore character name or item name, use *-ANY-*.
        - ♦ __Usage example:__ **OnGrab -ANY-,0,-1,-1,Bandages** will trigger the conversation when any character from army 0 grabs an item sack anywhere on the battlefield, just so long as the sack contains Bandages.
        - ♦ __Usage example:__ **OnGrab Tremolo Phalanges,-1,-1,8,-ANY-** will trigger the conversation when Tremolo Phalanges grabs any item sack that is 8 spaces from the left edge of the map.
9. **OnCharAttacked** – displays some dialog right before an attack is executed. This takes either two or three parameters, depending on whether you want to trigger dialog when a specific named character is attacked, or when a class of character or object is attacked. To trigger dialog when a unique, named character is attacked, use two parameters: turn

number and character name, including spaces. (This is what you'll use for boss taunts: "You think you can hurt me? Laughable!" and such.) To trigger dialog when a certain type of object is attacked, use three parameters: turn number, the word *None*, and the triggering className. (If you want the dialog to trigger on any turn where the specified character or class of character is attacked, make sure to use -1 for the turn number parameter!)

- There are optional fourth and fifth parameters as well: attacker name (if you want the conversation to only trigger if a certain character attacks the target); and non-attacker names delimited by colons (the names of characters who will ensure this dialog won't trigger if they are the ones to attack the target). You can use *-ANY-* for the fourth parameter if it can be any character that attacks the target, subject only to the exclusion list in the fifth parameter. You can also use *-ANY-* for the second parameter if any character can be attacked to trigger the dialog.
  - ♦ Usage example: **OnCharAttacked -1,None,Chair** will trigger the conversation as soon as a player clicks to attack a chair (it doesn't matter which chair); **OnCharAttacked -1,General Vile** will trigger the conversation as soon as the player clicks to launch an attack against General Vile.
  - ♦ Usage example: **OnCharAttacked -1,-ANY-,,Sarn Kamina** will trigger the conversation as soon as any character is attacked by a character named Sarn Kamina.
  - ♦ Usage example: **OnCharAttacked -1,General Vile,,-ANY-,Sarn Kamina:Emma Strider** will trigger the conversation as soon as a character named General Vile is attacked by *anyone other than* Sarn Kamina or Emma Strider.

10. **OnCharDeath** – this is what you'll use for death monologues: "Lo, I have been slain!" and such. Takes either two or three parameters, depending on whether you want to trigger dialog upon the death of a specific named character, or on the destruction of a class of character or object. To trigger dialog on the death of a unique, named character, use two parameters: turn number and character name, including spaces. To trigger dialog on the destruction of an object, use three parameters: turn number, the word *None*, and the triggering className. (If you want the dialog to trigger on any turn where the specified character or class of character dies, make sure to -1 for the turn number parameter!)
    - There are optional fourth and fifth parameters as well: killer name (if you want the conversation to only trigger if a certain character killed the target); and non-killer names delimited by colons (the names of characters who will ensure this dialog won't trigger if they are the ones to kill the target). You can use *-ANY-* for the fourth parameter if it can be any character that kills the target, subject only to the exclusion list in the fifth parameter. You can also use *-ANY-* for the second parameter if any character can be killed to trigger the dialog.
      - ♦ Usage example: **OnCharDeath -1,None,Barrel** will trigger the conversation as soon as a barrel is destroyed; **OnCharDeath -1,Jim Francis** will trigger the conversation as soon as the character named Jim Francis is killed.
      - ♦ Usage example: **OnCharDeath -1,General Vile,,Sarn Kamina** will trigger

the conversation as soon as a character named General Vile is killed a character named Sarn Kamina.

♦ Usage example: **OnCharDeath -1,General Vile,,-ANY-,Sarn Kamina:Emma Strider** will trigger the conversation as soon as a character named General Vile is killed by *anyone other than* Sarn Kamina or Emma Strider.

11. **OnLevel** – this triggers a conversation when a character levels up. Takes two parameters: the name of a character, and the level they must be reaching to trigger the dialog. Use *-ANY-* for the first parameter if it applies to any character leveling up; and use *-1* for the second parameter if it doesn't matter what level they're reaching.
12. **OnStat** – takes four parameters: the name of a character, the name of a stat, a mode of comparison, and a numerical amount. This checks to see if the value of the character's stat falls within the range indicated by the mode of comparison and the amount; if it does, the conversation is triggered. You can use *-ANY-* in lieu of the character name if any character should be able to trigger the dialog. The stat name can be any of the stat names listed under **IfStatGoTo** below. The comparison mode can be any of the following:
    - = (check for an exact match between stat value and amount)
    - g (check if stat value is greater than amount)
    - l (check if stat value is less than amount)
    - g= (check if stat value is greater than or equal to amount)
    - l= (check if stat value is less or equal to than amount)
        - ♦ Usage example: **OnStat Jim Francis,Level,=,20** will trigger the conversation as soon as it detects that Jim Francis is level 20.
13. **OnVal** – takes three parameters: the name of a preexisting custom variable, a mode of comparison, and a numerical amount. This checks to see if the value of the custom variable falls within the range indicated by the mode of comparison and the amount; if it does, the conversation is triggered. The comparison mode can be any of the following:
    - = (check for an exact match between variable value and amount)
    - g (check if variable value is greater than amount)
    - l (check if variable value is less than amount)
    - g= (check if variable value is greater than or equal to amount)
    - l= (check if variable value is less or equal to than amount)
        - ♦ Usage example: **OnVal Reputation,g=,10** will trigger the conversation if it detects that the custom variable Reputation exists and has a value of greater than or equal to 10.
14. **OnCharSelect** – it can use up to two optional parameters: Turn Number and Character Name, including spaces. If no parameters are specified, it will trigger the conversation as soon as the player selects any character.
15. **OnMoveComplete** – it can use up to two optional parameters: Turn Number and Character Name, including spaces. If no parameters are specified, it will trigger the conversation the first time any character finishes moving to any space.
    - ♦ Usage example: **OnMoveComplete -1,Ghost Soldier** will trigger the

conversation as soon as any character named “Ghost Soldier” finishes moving to a new space; the -1 Turn Number value tells the game that this can happen on any turn and still trigger the conversation.

16. **OnAttackSelect** – it can use up to three optional parameters: Turn Number; Character Name, including spaces; and Attack Name. Use *-1* if it doesn't matter which turn they select the attack on, *-ANY-* if it doesn't matter which character selects the attack, or *-ANY-* if it doesn't matter which attack they select. If no parameters are specified, it will trigger the conversation as soon as any character selects any attack.

- **Repetitions**
  1. By default, a conversation can only be triggered a single time per scene; after it's run, the conversation will not trigger again.
  2. To make a conversation repeatable, open up the XML file containing your dialog and add an *r* attribute to the very first branch of the conversation. Set it to the number of times you want the conversation to be able to repeat. If you want the conversation to be repeated without limit, set *r* to any integer less than 1 (0, -1, etc.)
     - ♦ Usage example: **<Dialog branch="0" r="3">** will allow the player to trigger that conversation up to three times during the scene.
- **Name**
  1. By default, dialog branches do not have names. However, you can name particular dialog branches for use with the *NamedBranch* reply type and *BNAME[]* special character.
  2. To name a dialog branch, open up the XML file containing your dialog and add a *name* attribute to the branch you want named, then type a string between the quotation marks.
     - ♦ Usage example: **<Dialog branch="23" name="Accept Bribe">** will allow the player to visit this dialog branch using the name *Accept Bribe* even if this branch's position in the conversation changes due to later edits.
- **Types of Script Actions**
  1. **AddObjective** – this adds a new objective to the current battle. AddObjective has one parameter: the text of the new objective.
  2. **AddObjectiveReticle** – this adds a visual indicator to whatever space you specify on the battlefield. AddObjectiveReticle has three parameters: Y coordinate, X coordinate, and Color in hexidecimal format.
     - ♦ Usage example: **AddObjectiveReticle 4,2,0xFFFFFF** will place a white objective reticle onto the battlefield 4 spaces from the top and 2 spaces from the left.
  3. **AddAttackerPortrait** – this adds the default character portrait of the current attacking character to the screen. One parameter: facing (right or left).
     - Note: this action only works in battles, and only when a character is in the middle of attacking (as one will be with the OnCharDeath and OnCharAttacked triggers). If no attacking character is found, no portrait will be shown.
  4. **AddPortrait** – this adds a character portrait to the screen. AddPortrait has three required parameters: Portrait Name, Portrait Filename, and Facing. Portrait Name can be anything you want; this will be the word or phrase you use to move, flip, or remove the portrait later; Portrait Filename must be the name of the image file, minus the *.png* file

extension (it will be loaded from *Data > Characters > _Portraits*); and Facing must be either *right* or *left*, indicating the direction the character portrait will face.

- In addition to the required parameters, there are three additional, optional parameters: Palette, Plus Y and Plus X. Palette can be set to any army color (for available army colors, see **Set Army Color** in Conditions); if not set to any valid army color, the portrait will use the Violet army palette. Plus Y and Plus X each take an integer value; they shift the portrait's starting position the designated number of pixels vertically and horizontally, respectively.
    - ♦ Usage example: **AddPortrait Sarn Kamina,Lizardman_F,right,Red,0,150** will load *Lizardman_F.png* as a new portrait under the name *Sarn*, and the portrait will face right. It will be positioned 150 pixels to the right of its default position.

5. **AddSpeakerPortrait** – this adds the default character portrait of the current speaking character to the screen. One parameter: facing (right or left).
    - Note: this action only works in battles, and only when the speaking character is present on the battlefield. If no such character is found, no portrait will be shown.
6. **AddStatus** – this attempts to give a status effect to a named character. There are three parameters: the Character's Name including spaces; the Status Effect to give; and the Elemental Resistance that applies.
    - Note: to have a status effect take hold automatically without checking character resistance, use *AutoAdd* as the string value for the elemental resistance parameter.
    - You can use any status effect in the game with this action; see the full list on pp. 12-13.
        - ♦ Usage example: **AddStatus Claude Mann,Burning,Heat** will check for a character named Claude Mann—if it finds him, it will use his Heat resistance to determine if he gains Burning status, just as if he were hit with a Heat attack.
        - ♦ Usage example: **AddStatus Claude Mann,Burning,AutoAdd** will check for a character named Claude Mann—if it finds him, he will gain Burning status automatically regardless of his resistances.
7. **AddStatusAt** – this attempts to give a status effect to a character at a specified set of battlefield coordinates; if it finds a character at those coordinates, it then checks to see if the character gains the named status effect as if it it had been the target of a skill. There are four parameters: the Y coordinate, the X coordinate, the Status Effect to give, and the Elemental Resistance that applies.
    - Note: to have a status effect take hold automatically without checking character resistance, use *AutoAdd* as the string value for the elemental resistance parameter.
        - ♦ Usage example: **AddStatusAt 6,4,Burning,Heat** will check for a character at the coordinates 6 , 4—if it finds one, it will use the character's Heat resistance to determine if he/she gains Burning status, just as if the character were hit with a Heat attack.
        - ♦ Usage example: **AddStatusAt 6,4,Burning,AutoAdd** will check for a character at the coordinates 6 , 4—if it finds one, that character will gain Burning status automatically.

8. **AddTag** – this adds a new character tag to a unique, named character or object on the battlefield. There are at least two parameters: Character Name and Tag Type, plus an extra parameter for each tag parameter required for use with that Tag Type. (See **B. Characters and Objects** for more about tags.)
9. **AddTagToArmy –** this adds a new character tag to every character in a designated army. There are at least two parameters: Army Number and Tag Type, plus an extra parameter for each tag parameter required for use with that Tag Type. (See **B. Characters and Objects** for more about tags.)
   - ♦ Usage example: **AddTagToArmy 1,Passive** will cause all of army 1 to adopt a passive AI approach.
   - ♦ Usage example: **AddTagToArmy 2,TargetValue,0.5** will halve the target value of all characters in Army 2, making them less likely to be targeted for skills and attacks by the computer AI relative to characters of other armies.
10. **AddTextOverlay** – this adds a text overlay at the top of the screen. There is one parameter: the text to be contained in the overlay.
    - Note: there is a second, optional parameter: number of frames that the text overlay is to remain onscreen before automatically vanishing. If you use this parameter, delimit it with a double colon (**::**), *not* a comma. If you do not use the second parameter, the text overlay will remain onscreen until the turn ends or a second text overlay replaces it.
      - ♦ Usage example: **AddTextOverlay Defeat General Vile!** will cause a text overlay to appear at the top of the screen with the text "Defeat General Vile!"
      - ♦ Usage example: **AddTextOverlay Warning: enemy approaching!::135** will cause a text overlay to appear at the top of the screen with the text "Warning: enemy approaching!" The text will automatically vanish after 135 frames (about 3 seconds).
11. **AssignSpeakerAtCoords** – this moves the camera to a specified set of battlefield coordinates; if it finds a character at those coordinates, it then highlights that character as if that character were the speaker. There are two parameters: the Y coordinate, and the X coordinate.
    - This is recommended for dialog among specific enemy characters without unique names.
12. **ChangeCharClass** – this changes the class name used by the named character, as well as changing its sprite. There are three parameters: the Character's Name, including spaces; the name of the Class as it is displayed in the game; and the name of the Sprite Sheets you want to start using.
    - ChangeCharClass is used to accomplish unit class promotions, or for a more dramatic class change. It will work only if the character is present on the battlefield!
      - ♦ Usage example: **ChangeCharClass Claude Mann,Paladin,Mantis Knight** will cause the character named Claude Mann to become a "Paladin" that uses Mantis Knight sprite sheets.
13. **ChangeCondition** – any battlefield condition you can set with a condition tag, you can change using this Action. Use the same parameters as you would defining the condition initially.

- ♦ Usage example: **ChangeCondition Weather,Rain** will cause the map's weather to change to rain.
- ♦ Usage example: **ChangeCondition Fog of War,true** will turn on fog of war.

14. **CheckForMoreDialog** – this checks for more dialog queued up, then ends the conversation immediately if there isn't. (See the EndConvImmediately action below; compare with the reply type by the same name.) No parameters.
    - CheckForMoreDialog is used with character death monologues, since multiple characters with their own death monologues can suffer mortal wounds in a single attack.
15. **ClearInv** – removes *all* items from a character's inventory. There is one parameter: the Name of the Character whose inventory you want to clear, including spaces.
16. **ClearLevelUpAttacks** – clears *all* attacks that a character is set to learn upon leveling up in the future. There is one parameter: the name of the character whose list of level-up attacks you want to clear, including spaces.
17. **ClearObjectiveReticle** – this removes the objective reticles (*see* **AddObjectiveReticle**) from whatever space you specify on the battlefield. ClearObjectiveReticle has two parameters: Y coordinate and X coordinate.
18. **ClearPortraits** – this removes all character portraits from the screen. No parameters.
19. **DamageChar** – this deals damage directly to a named character. The three parameters are: Character Name, Damage Amount, and Element (Slash, Pierce, Heat, Cold, etc.) If you don't want the character's resistance to certain elements to impact the total damage they receive, just use something that isn't a normal element for the third parameter, like "Banana" or something.
20. **DamageCharAt** – this deals damage directly to a character at specific coordinates. Exactly like **DamageChar**, but instead of using a character name, it uses coordinates. Four parameters: Y coordinate, X coordinate, Damage Amount, and Element.
    - ♦ Usage example: **DamageCharAt 4,12,10,Banana** will deal 10 base Banana damage to whatever character is 4 rows down from the top of the map and 12 columns right of the left side of the map. Because "Banana damage" is not a thing in Telepath Tactics, the damage will not be resisted.
    - ♦ Usage example: **DamageCharAt -Y-,-X-,8,Pierce** will deal 8 base Pierce damage (i.e. damage that will be reduced by the target's Pierce Resistance) to whatever character triggered the dialog or script, since the game will just fill in that character's coordinates due to the use of *-Y-* and *-X-*.
21. **DelArr** – this deletes an entry in a custom array, causing all of the entries after it in the chosen array to collapse forward one position. DelArr uses two parameters: Array Name (a string: the name of the array to delete an entry from), and Delete from Position (*front*, *end*, *all*, or a specific integer reflecting the position of the entry to delete).
22. **DismissChar** – this removes a character from the player's army roster. There is one parameter: the name of the character to dismiss, with a colon delimiting the first and last name. (Make sure you use a colon and not a space!)
    - The reason the name is delimited here rather than typed out with spaces is because the game looks directly into the *CharClasses.xml* file when it spawns characters on your team; where you see a forward slash separating character names in the

CharClasses.xml file, just use a colon instead.

♦ Usage example: **DismissChar Claude:Mann** will cause the character named Claude Mann to be removed from the player's army roster.

23. **EndBattle** – immediately take the player to the victory screen.
24. **EndConvImmediately** – this is a very important action with a very specific purpose: it allows you to run scripts without ever displaying character dialog to the player! EndConvImmediately closes the dialog screen immediately, without waiting for the player to click. There are no parameters.
    - If you ever want scripts to be run at the end of a conversation, after the player has clicked the final reply, you *must* use a NewBranch reply and direct it to a dummy dialog branch containing any actions you want to run, plus an EndConvImmediately action.
        - ♦ Usage example: Suppose we want to have a character leave the battlefield right after the end of a conversation. We can't just use an EndConv reply and a RemoveChar action on the last branch to accomplish this, since the action will run as soon as the branch is reached, causing the character to leave prematurely. Instead, we must add one extra branch with RemoveChar and EndConvImmediately, then use a NewBranch reply to direct us there. From the player's perspective, the character will leave and the conversation will end simutaneously the moment he clicks!
25. **EnemiesLeft** – counts the enemies remaining on the battlefield for a particular army, then automatically creates and sets a custom val named *_EnemiesLeft* equal to that number. One parameter: army number (the number denoting the army whose enemies the game must count).
    - Optional second parameter: enemy army number (will limit the counting to only characters belonging to that specific enemy army).
        - ♦ Usage example: **EnemiesLeft 0** will count how many enemies remain on the battlefield for the player.
26. **FadePortrait** – causes a portrait to fade out or fade in. Has two parameters: Portrait Name and Number of Frames. Portrait Name is just the Portrait Name you used when running AddPortrait (this tells the game which portrait to flip). Number of Frames tells the game how many frames to take fading the portrait in or out. There are two additional optional parameters: Delay and Mode. Delay is the number of frames to wait before beginning to fade the portrait in or out. Mode is either *out* or *in*; this specifies whether to fade the portrait out or fade it in. (By default, portraits fade out with 0 frames of delay.)
    - ♦ Usage example: **FadePortrait Jim,45,10** will cause the portrait named Jim to fade out over 45 frames (i.e. 1 in-game second) after waiting 10 frames.
    - ♦ Usage example: **FadePortrait Emma,30,0,in** will cause the portrait named Emma to go transparent and then fade in over the span of 30 frames.
27. **FadePortraits** – just like FadePortrait, but instead of acting on a specific portrait, it causes *all* portraits to fade out or fade in at once. Has one parameter: Number of Frames; this tells the game how many frames to take fading the portraits in or out. There are two additional optional parameters: Delay and Mode. Delay is the number of frames to wait before beginning to fade the portraits in or out. Mode is either *out* or *in*; this

specifies whether to fade the portraits out or fade them in. (By default, portraits fade out and with 0 frames of delay.)

- ♦ __Usage example__: **FadePortraits 90** will cause all portraits in the scene to immediately begin fading out over the span of 90 frames (i.e. 2 in-game seconds).

28. **FlipPortrait** – flips a right-facing portrait so it faces left, or vice versa. Has two parameters: Portrait Name and X Offset. Portrait Name is just the Portrait Name you used when running AddPortrait (this tells the game which portrait to flip). X Offset is an optional parameter that repositions the portrait when it's flipped; you can use this to avoid jarring results when flipping an uncentered portrait.
29. **FlushDeathQueue** – when a conversation is triggered by OnCharDeath, the character(s) or object(s) whose death triggered the conversation will remain on the battlefield. Use this action to remove any characters or objects queued up for death from the battlefield.
    - Note: do *not* use this action unless and until you are certain that any dying characters involved in the conversation won't have any further lines of dialog!
30. **ForArr** – runs a *for loop* based upon a range of entries within a custom array. In each step of the loop, it automatically runs GetArr to retrieve the current position in the array, then runs a named script. ForArr has five parameters: Array Name; Start Position (*front*, *end*, or a specific integer reflecting an entry position within the array); End Position (*front*, *end*, or a specific integer reflecting an entry position within the array); Increment By (an integer value by which to increment the current position at each step of the loop); and Script Name (the name of the script to run at the conclusion of each step in the loop).
31. **GetArr** – retrieves a specific entry within a custom array, then sets either the custom string *_ArrStr* (if it's a string) or the custom value *_ArrVal* (if it's a value) equal to it. GetArr has three parameters: Array Name (a string containing the name of the custom array to retrieve the entry from), Get from Position (*front*, *end*, or a specific integer reflecting the position of the entry to retrieve), and Get As (*auto*, *STR*, or *VAL*; if *auto*, the game will make its best guess as to whether the entry is a string or a numerical value).
32. **GetCharsDist** – finds the total distance in spaces between two characters, then sets a custom val named *_Dist* equal to that number. Two parameters: Name of Character 1, and Name of Character 2.
33. **GetCharSpaceDist** – finds the total distance in spaces between a character and a chosen space on the battlefield, then sets a custom val named *_Dist* equal to that number. Three parameters: the Name of the Character, the Y Coordinate of the space, and the X Coordinate of the space.
34. **GetItemValue** – finds the *itemValue* of the named item, then sets a custom val named *_ItemValue* equal to that number. One parameter: item name.
35. **GiveExp** – gives a named character a specified amount of experience points. There are two parameters: the Character's Name, including spaces; and the Amount of Experience points to give that character.
    - If the character's experience points go over 100, the character will level up right then

and there.
- If the character is not present on the battlefield, nothing will happen using this action.

36. **GiveItem** – spawns an item directly in the named character's inventory. There are two parameters: the Character's Name, including spaces; and the Name of the Item to spawn. If the item is equippable and you want the character to auto-equip the item, include a third parameter value: *equip*. If you want to spawn the item in the Common Inventory instead of in the inventory of a particular character, use *Common Inventory* as the value of the first parameter.
    - Note: if you call **GiveItem** during a static cut scene, it will not work unless you use *Common Inventory* as the value of the first parameter.
        - ♦ Usage example: **GiveItem Common Inventory,Bandages** will spawn Bandages in the Common Inventory.
        - ♦ Usage example: **GiveItem Emma Strider,Steel Blade,equip** will spawn a Steel Blade in Emma Strider's inventory and equip her with it.
37. **GoTo** – immediately switch to a different branch in the conversation. There is only one parameter: the Branch to Go to.
38. **GoToLastBranch** – immediately switch to whatever branch the player was at in the conversation before the current branch. There is one optional parameter: Number of Branches Back in the conversation history to travel.
39. **IfGoneGoTo** – switch to a different conversation branch if a certain named character is not present on the battlefield. There are two parameters to the IfGoneGoTo action: name of the character to check for, including spaces; and branch.
    - Note: if this action returns true, the dialog will proceed to the named branch *immediately*, skipping any remaining actions in this branch!
        - ♦ Usage example: **IfGoneGoTo Jim Francis,2** will switch the conversation to branch 2 if the game detects that no character named Jim Francis is present on the battlefield.
40. **IfGoneRun** – exactly like IfGoneGoTo, but runs a script instead of going to a new branch. The parameters are identical to those in IfGoneGoTo except for the last, which should be the name of the script to run instead of the branch to go to.
    - Note: unlike with the "GoTo" variant," if this action returns true, the remaining actions in the branch *will* still run.
41. **IfItemGoTo** – switch to a different conversation branch if a certain item is present in one or more characters' inventories. There are four parameters to the IfItemGoTo action: Name of the Character whose inventory the game will check, Name of the Item to check for, whether the item Has to Be Equipped, and the Branch to Go to if the item is found.
    - You can use *Anyone* for the character name parameter to have the game check the inventories of every character in Army 0.
    - Note: if this action returns true, the dialog will proceed to the named branch *immediately*, skipping any remaining actions in this branch!
        - ♦ **Usage example: IfItemGoTo Jim Francis,Binoculars,false,2** will switch the conversation to branch 2 if the game detects that Jim Francis is present on the battlefield and has Binoculars in his inventory.

- ♦ <u>Usage example</u>: **IfItemGoTo Anyone,Fire Sword,true,12** will switch the conversation to branch 12 if the game detects that any character in Army 0 has a Fire Sword equipped.

42. **IfItemRun** – exactly like IfItemGoTo, but runs a script instead of going to a new branch. The parameters are identical to those in IfItemGoTo except for the last, which should be the name of the script to run instead of the branch to go to.
    - <u>Note</u>: unlike with the "GoTo" variant," if this action returns true, the remaining actions in the branch *will* still run.
43. **IfOnCoordsRun** – checks to see if characters are on a particular space during a conversation; if a match is found, the game runs a script. This action has four parameters: Character Name, Y coordinate, X coordinate, and Script Name. If the named character is at the battlefield coordinates you specified, the named script is run.
    - <u>Note</u>: you can also specify *NO_ONE* for the first parameter, in which case the game checks to see if the space is entirely empty--if it is, then the named script is run; or *ANY_ONE*, in which case it'll count any character on the space.
        - ♦ <u>Usage example</u>: **IfOnCoordsRun NO_ONE,6,4,Spawn At 6 4** will run the script named *Spawn At 6 4* if it detects no characters on the space 6 down from the top of the map and 4 right of the left edge of the map.
44. **IfPathClearRun** – checks to see if a character has a clear movement path to a particular space during a conversation; if so, the game runs a script. This action has four parameters: Character Name, Y coordinate, X coordinate, and Script Name. If the named character has a path to the battlefield coordinates you specified, the named script is run.
    - ♦ <u>Usage example</u>: **IfPathClearRun Emma Strider,6,4,Move To 6 4** will run the script named *Move To 6 4* if it detects that the character named Emma Strider has a clear path to move to the space 6 down from the top of the map and 4 right of the left edge of the map.
45. **IfStatGoTo** – switch to a different conversation branch if one of a character's stats has a certain value. There are five parameters to the IfStatGoTo action: Character Name with spaces, Stat Name, mode of comparison, amount, and branch. Character name tells the game whose stats to look at; stat name tells the game which stat to compare; mode of comparison tells the game how to compare the stat's value to the amount; amount tells it what to compare the custom variable to; and branch tells it which conversation branch to switch to if the game finds a match.
    - The comparison mode can be any of the following:
        - = (check for an exact match between variable value and amount)
        - g (check if variable value is greater than amount)
        - l (check if variable value is less than amount)
        - g= (check if variable value is greater than or equal to amount)
        - l= (check if variable value is less or equal to than amount)
    - The stat name can be any of the following:
        - *Army* – the army number of the army the character belongs to (0 for the player's army, 1 for the enemy, etc.)
        - *Level*

- *Exp* - experience points.
- *Damage* - health lost.
- *Health* - current health.
- *Max Health* *
- *Drain* - energy below maximum.
- *Energy* - current energy.
- *Max Energy* *
- *Strength* *
- *Psy Power* *
- *Psy Defense* *
- *Steps Left* - steps remaining that character can move.
- *Speed* *
- *Accuracy* *
- *Dodge* *
- *Counter Limit* * - maximum number of counterattacks a character can perform per turn.
- *Perception* * - spaces the character can see through fog of war.
- *Pierce Res.* *
- *Slash Res.* *
- *Crush Res.* *
- *Mental Res.* *
- *Heat Res.* *
- *Cold Res.* *
- *Light Res.* *
- *Shadow Res.* *
- *Poison Res.* *
- *Done* - 0 if a character can still act, 1 if it can't. (If it's another player's turn, then this is a legacy value from the character's owner's last turn.)
- *Attacked* - 0 if character hasn't launched an attack of type "EndTurn" or "CanMove" since start of character's last turn, and 1 if character has. ("Unlimited" and “UseOnce” attacks don't affect this stat.)
- *Countered* - number of counterattacks launched since start of character's last turn.
- *Y Coord*
- *X Coord*
- *MoveType* – **a string**: use *land, swimming* or *flying* in lieu of the mode of comparison parameter, and leave the amount parameter blank.
- *Direction* – **a string**: use *Up, Down, Left, Right* or *None* in lieu of the mode of comparison parameter, and leave the amount parameter blank.
- *Name* – **a string**: use *the character's full name (with spaces) to match against* in lieu of the mode of comparison parameter, and leave the amount parameter blank.

- *FirstName* – **a string**: use *the character's first name to match against* in lieu of the mode of comparison parameter, and leave the amount parameter blank.
- *LastName* – **a string**: use *the character's last name to match against* in lieu of the mode of comparison parameter, and leave the amount parameter blank.
- *Class* – **a string**: use *the character class to match against* in lieu of the mode of comparison parameter, and leave the amount parameter blank.
- *Sprite* – **a string**: use *the sprite type to match against* in lieu of the mode of comparison parameter, and leave the amount parameter blank.
- *Portrait* – **a string**: use *the portrait name to match against* in lieu of the mode of comparison parameter, and leave the amount parameter blank.
- *Race* – **a string**: use *the race to match against* in lieu of the mode of comparison parameter, and leave the amount parameter blank.
- *Sex* – **a string**: use *the sex to match against* in lieu of the mode of comparison parameter, and leave the amount parameter blank.
  - For any stat listed above with an asterisk on the end, you can append *BASE* with a space to the front of it to get the stat's base value (i.e. ignoring temporary modifiers from items or other buffs).
  - Note: if this action returns true, the dialog will proceed to the named branch *immediately*, skipping any remaining actions in this branch!
  - ♦ Usage example: **IfStatGoTo Claude Mann,Strength,l,10,2** will switch the conversation to branch 2 if it detects that the character named Claude Mann is present on the battlefield and has a Strength of less than 10.
  - ♦ Usage example: **IfStatGoTo Claude Mann,BASE Strength,g,5,7** will switch the conversation to branch 7 if it detects that the character named Claude Mann is present on the battlefield and has a Strength greater than 5, ignoring any items or other effects which may have given him a temporary Strength boost / penalty.
  - ♦ Usage example: **IfStatGoTo Claude Mann,MoveType,swimming,,6** will switch the conversation to branch 6 if Claude Mann's MoveType is *swimming*.
  - ♦ Usage example: **IfStatGoTo Claude Mann,Class,Swordsman,,9** will switch the conversation to branch 9 if Claude Mann's class is *Swordsman*.

46. **IfStatRun** – exactly like IfStatGoTo, but runs a script instead of going to a new branch. The parameters are identical to those in IfStatGoTo except for the last, which should be the name of the script to run instead of the branch to go to.
    - Note: unlike with the "GoTo" variant," if this action returns true, the remaining actions in the branch *will* still run.
47. **IfStringGoTo** – switch to a different conversation branch if a string custom variable has a certain value. There are three parameters to the IfStringGoTo action: Variable Name, String and Branch. Variable Name tells the game what custom string variable to check; String tells it what to look for; and Branch tells it which conversation branch to switch to if the game finds a match.
    - Note: if this action returns true, the dialog will proceed to the named branch

immediately, skipping any remaining actions in this branch!

    - ♦ Usage example: **IfStringGoTo ChoseTraining,Mental,4** will switch the conversation to branch 4 if it detects that the custom string variable *ChoseTraining* is currently set to *Mental*. (See **SetString**.)

48. **IfStringRun** – exactly like IfStringGoTo, but runs a script instead of going to a new branch. The parameters are identical to those in IfStringGoTo except for the last, which should be the name of the script to run instead of the branch to go to.
    - Note: unlike with the "GoTo" variant," if this action returns true, the remaining actions in the branch *will* still run.
49. **IfTagGoTo** – switch to a different conversation branch if the named character has a certain tag. There are four parameters to the IfTagGoTo action: Character Name with spaces, Tag Name, Tag Parameters (if any), and Branch. The tag name can be anything listed in **Characters and Destructible Objects > Using Character Tags**, or even custom tags you've created yourself.
    - Note: If you put in any text for the Tag Parameters parameter, this action will only return true if the both the tag name *and* tag parameters all match.
    - Note: if this action returns true, the dialog will proceed to the named branch *immediately*, skipping any remaining actions in this branch!
        - ♦ Usage example: **IfTagGoTo Claude Mann,Promoted,,7** will switch the conversation to branch 7 if the character Claude Mann has the *Promoted* tag.
50. **IfTagRun** – exactly like IfTagGoTo, but runs a script instead of going to a new branch. The parameters are identical to those in IfTagGoTo except for the last, which should be the name of the script to run instead of the branch to go to.
    - Note: unlike with the "GoTo" variant," if this action returns true, the remaining actions in the branch *will* still run.
51. **IfValGoTo** – switch to a different conversation branch if a numerical custom variable has a certain value. There are four parameters to the IfValGoTo action: Variable Name, Mode of Comparison, Amount, and Branch. Variable Name tells the game what custom variable to compare; Mode of Comparison tells the game how to compare the variable's value to the amount; Amount tells it what to compare the custom variable to; and Branch tells it which conversation branch to switch to if the game finds a match. The comparison mode can be any of the following:
    - = (check for an exact match between variable value and amount)
    - g (check if variable value is greater than amount)
    - l (check if variable value is less than amount)
    - g= (check if variable value is greater than or equal to amount)
    - l= (check if variable value is less or equal to than amount)
        - Note: if this action returns true, the dialog will proceed to the named branch *immediately*, skipping any remaining actions in this branch!
        - ♦ Usage example: **IfValGoTo Money,l,100,2** will switch the conversation to branch 2 if it detects that the custom variable Money exists and has a value of less than 100. (See **SetVal**.)
52. **IfValRun** – exactly like IfValGoTo, but runs a script instead of going to a new branch.

The parameters are identical to those in IfValGoTo except for the last, which should be the name of the script to run instead of the branch to go to.

- Note: unlike with the "GoTo" variant," if this action returns true, the remaining actions in the branch *will* still run.

53. **IfValsGoTo** – this is *exactly* like IfValGoTo above, but it compares two custom variables instead of comparing a single variable against a fixed value. There are four parameters to the IfValsGoTo action: variable name, mode of comparison, second variable name, and branch.
    - Note: if this action returns true, the dialog will proceed to the named branch *immediately*, skipping any remaining actions in this branch!
        - ♦ Usage example: **IfValsGoTo Money,g=,Cost,5** will switch the conversation to branch 5 if it detects that the custom variable Money exists and has a value greater than or equal to a second custom variable, Cost.
54. **IfValsRun** – exactly like IfValsGoTo, but runs a script instead of going to a new branch. The parameters are identical to those in IfValsGoTo except for the last, which should be the name of the script to run instead of the branch to go to.
    - Note: unlike with the "GoTo" variant," if this action returns true, the remaining actions in the branch *will* still run.
55. **ItemDrop** – drops a single item sack onto a randomly chosen space on the battlefield. Has one or more parameters: the Name(s) of the Item(s) contained in the item sack.
    - Note: Just as with item drops which occur in multiplayer, the game will be more likely to choose spaces that are closer to being midway between the armies' spawn locations, and will not use spaces within a certain minimum distance of any character's spawn point. If no suitable spaces exist, the item drop will not occur.
    - Note: If you provide parameters that do not match existing item names, the game will just select items at random from the full list of available items, weighted by their commonality parameters.
56. **KillChar** – this kills the named character instantly. There is one parameter: the name of the character to kill. (Compare with RemoveChar, below.)
        - ♦ Usage example: **KillChar Jim Francis** will cause the character named Jim Francis to die.
57. **KillCharAt** – this kills a named character at a specific set of coordinates instantly. There are two parameters: Y coordinate and X coordinate.
        - ♦ Usage example: **KillCharAt 3,4** will kill whatever named character currently occupies the space 3 rows down from the top and 4 columns right from the left side of the battlefield.
58. **LightBackground** – lights the background picture in a static cut scene (*see **Section H*** below). Has one parameter: Lighting Preset. This tells the game how to light the portrait; you can use any of the same presets that are used with the Global Lighting condition.
    - Note: If you leave the parameter blank (or use "Custom" for it), you can use optional second, third, and fourth parameters: Red, Green, and Blue values for custom lighting. This works just the same as with the Global Lighting condition.
59. **LightPortrait** – lights a portrait with global lighting. Has two parameters: Portrait Name and Lighting Preset. Portrait Name is just the Portrait Name you used when

running AddPortrait (this tells the game which portrait to light). Lighting Preset tells the game how to light the portrait; you can use any of the same presets that are used with the Global Lighting condition.

- Note: If you leave the second parameter blank (or use "Custom" for it), you can use optional third, fourth, and fifth parameters: Red, Green, and Blue values for custom lighting. This works just the same as with the Global Lighting condition.

60. **Lock** – this looks for a door at a specified set of battlefield coordinates; if it finds a door at those coordinates, it closes and locks it. There are two parameters: the Y coordinate, and the X coordinate.
61. **LoseBattle** – immediately take the player to the defeat screen.
62. **MoveCam** – instead of focusing on the speaker, this moves the camera to a specified set of battlefield coordinates during a given branch of dialog. There are two parameters: the Y coordinate, and the X coordinate. There is also an optional third parameter, Frames —a positive integer telling the game how many frames it should take to move the camera to its destination. (By default, it's 30 with this action.)
    - ♦ Usage example: **MoveCam 5,8** will focus the camera on the space 5 rows down from the top and 8 columns right from the left side of the battlefield, and take 30 frames to do it.
    - ♦ Usage example: **MoveCam 5,8,90** will focus the camera on the space 5 rows down from the top and 8 columns right from the left side of the battlefield, but will take 90 frames to do so.
63. **MoveChar** – this moves the named character to coordinates of your choice. There are three parameters: the Name of the Character to move, including spaces; the Y coordinate to move to; and the X coordinate to move to.
    - ♦ Usage example: **MoveChar Jim Francis,5,8** will cause the character named Jim Francis to automatically move from his current position to the space 5 rows down from the top and 8 columns right from the left side of the battlefield.
64. **MoveDialogBox** – This repositions the dialog box vertically onscreen. It takes one parameter, Number of Pixels to move the dialog box.
65. **MovePortrait** – this moves a character portrait across the screen. MovePortrait has four required parameters: Portrait Name, Y to Move, X to Move, and Number of Frames. Y to Move determines the number of pixels to move vertically; X to Move determines the number of pixels to move horizontally; and Number of Frames is how many frames the portrait should take to complete moving.
66. **NewScene** – this immediately jumps to a new battle or cut scene. It has one parameter: the Name of the Scene to jump to (minus the .xml extension).
67. **NextScene** – this immediately jumps to the battle or cut scene specified in the current scene's *nextbattle* attribute. No parameters.
68. **OpenInv** – this opens the Common Inventory menu. It has one parameter: the name of the character currently accessing the Common Inventory. (If you want to display the entire team for purposes of swapping items into and out of the Common Inventory, use *WHOLE_TEAM* for the parameter value.)
69. **PlayAnim** – has a named character play a specified animation. There are two parameters: the Character's Name, including spaces; and the Name of the Animation.

- If the character is not present on the battlefield, nothing will happen using this action. If the character is present but does not have the named animation, the character will play his or her default skill animation instead.

70. **PlayLoop** – this plays a looping sound effect when the conversation reaches the current branch of dialog. There is one parameter: the name of the sound loop to play.
    - Use *None* as the parameter to silence any loops currently playing.
        - ♦ Usage example: **PlayLoop Crowd** will play the *Crowd* sound loop.
71. **PlayMusic** – this silences the music currently playing and begins playing a new track of your choice. There is one parameter: the name of the musical track to play.
    - Note: to simply silence whatever music is playing, use *None* as the parameter.
        - ♦ Usage example: **PlayMusic TacticiansDuel** will silence the current music track and begin playing the song "Tacticians' Duel."
        - ♦ Usage example: **PlayMusic None** will silence the current music track without playing any other music.
72. **PlaySound** – this plays a one-shot sound effect when the conversation reaches the current branch of dialog. There is one parameter: the name of the sound file to play, minus the mp3 extension.
    - ♦ Usage example: **PlaySound menuSelect** will play the *menuSelect.mp3* file within Data > Sounds.
73. **RecruitChar** – this adds a new character to the player's army roster. There is one parameter: the name of the character to recruit, with a colon delimiting the first and last name. (Make sure you use a colon and not a space!)
    - The reason the name is delimited here rather than typed out with spaces is because the game looks directly into the *CharClasses.xml* file when it spawns characters on your team; where you see a forward slash separating character names in the *CharClasses.xml* file, just use a colon instead.
        - ♦ Usage example: **RecruitChar Jennifer:Faust** will cause the character named Jennifer Faust to be added to the player's army roster.
74. **RemoveChar** – this removes the named character from the battlefield. There is one parameter: the Name of the Character to remove.
    - Characters removed with a RemoveChar action are not considered dead; they have merely left the battle, like actors obeying a direction to "Exit Scene." (Compare with KillChar, above.)
        - ♦ Usage example: **RemoveChar Jim Francis** will cause the character named Jim Francis to leave the battle.
75. **RemoveCharAt** – if a character is found at the coordinates specified, that character is removed from the battlefield (see RemoveChar above). There are two parameters: the Y coordinate, and the X coordinate.
    - ♦ Usage example: **RemoveCharAt 5,8** will cause any character or object on the space 5 rows down from the top and 8 columns right from the left side of the battlefield to leave the battle.
76. **RemoveConv** – removes the named conversation from the game's memory; it cannot be triggered again during this scene. The number of parameters varies: the first is always

the Type of Trigger used by the conversation to be removed, and the others are its Trigger Parameters.

77. **RemoveCurrConv** – removes the current conversation from the game's memory; it cannot be triggered again during this scene. No parameters.
78. **RemoveGlow** – removes the selection glow from whatever character currently has it. There are no parameters.
79. **RemoveItem** – unequips and removes an item from the named character's inventory. There are two parameters: the Character's Name, including spaces; and the Inventory Position of the item to remove. If you want to remove the item from the Common Inventory instead of in the inventory of a particular character, use *Common Inventory* as the value of the first parameter.
    - You can see the numerical position of an item by looking at a character's inventory: the item in the top-left is at position 0; the item to its right is at position 1; and so on.
    - Note: if you call **RemoveItem** during a static cut scene, it will not work unless you use *Common Inventory* as the value of the first parameter.
        - ♦ Usage example: **RemoveItem Jim Francis,0** will remove the very first item found in Jim Francis's inventory.
80. **RemoveItemByName** – exactly like **RemoveItem**, but takes the Item Name instead of inventory position for its second parameter. This action always removes the first item by that name that it finds in the character's inventory. If you want to remove the item from the Common Inventory instead of in the inventory of a particular character, use *Common Inventory* as the value of the first parameter.
    - Note: if you call **RemoveItemByName** during a static cut scene, it will not work unless you use *Common Inventory* as the value of the first parameter.
        - ♦ Usage example: **RemoveItemByName Jim Francis,Bandages** will remove the first item named “Bandages” that appears in Jim Francis's inventory.
81. **RemoveObjective** – this removes an existing objective from the current battle. RemoveObjective has one parameter: the text of the objective to remove. (If the text is not an exact match, the objective won't be removed.)
    - Note: you can use *-ALL-* for this action's parameter to remove all existing objectives from the battle.
82. **RemoveObjsAt** – remove all destructible objects (including bridges) found at the coordinates specified. There are two parameters: the Y coordinate, and the X coordinate.
83. **RemovePortrait** – this removes a character portrait to the screen. RemovePortrait has one parameter: Portrait Name (this must be identical to the name you used when you added the portrait with AddPortrait).
84. **RemoveReply** – this removes a reply option from a particular branch within the current conversation. Two parameters: Reply Text, and Branch Number. For this action, delimit the parameters with a double colon (**::**) instead of commas.
        - ♦ Usage example: **RemoveReply Want to buy my sword?::12** will look for the reply in branch 12 of the current conversation using the text “Want to buy my sword?” If it finds this reply in branch 12, it will remove it from that branch.
85. **RemoveSpawn –** this prevents a character from spawning on a future turn (but only if said character is going to spawn by virtue of having been painted onto the map; this *will*

not prevent a SpawnChar action from triggering). There is one parameter: the name of the character to keep from spawning.

- Note: this only works to keep a character from spawning *on a future turn*. If you use this action on a character who has already spawned, it will not remove them from the battlefield!
- You may use optional second and third parameters, Y Coordinate and X Coordinate, denoting the particular space the named character is due to spawn on. This will help the game find the right character if the character's name is shared by other units.
- You may use an optional fourth parameter, Turn to Spawn On, denoting the particular turn when the named character is due to spawn. This will help the game find the right character if the character's name is shared by other units.
  - ♦ Usage example: **RemoveSpawn Bloodbeard's:Bandit,1,2,5** will look for the character named Bloodbeard's Bandit who is scheduled to spawn at y coordinate 1 and x coordinate 2 on turn 5, and then ensure that he does not spawn.

86. **RemoveTag –** this removes a character tag from a unique, named character or object on the battlefield. There are two parameters: Character Name and Tag Type. (See B. Characters and Objects for more about tags.)
    - Note: Optionally, you can also include parameter values as additional parameters if the tag type uses parameters and you want to specifically remove the tag with those parameter values.
87. **RemoveTagFromArmy** – this removes a character tag from every character in a given army. There are two parameters: Army Number and Tag Type. (See B. Characters and Objects for more about tags.)
88. **RemoveTagFromCharAt –** this removes a character tag from a character at particular coordinates on the battlefield. There are three parameters: Y Coordinate, X Coordinate and Tag Type. (See B. Characters and Objects for more about tags.)
    - Note: Optionally, you can also include parameter values as additional parameters if the tag type uses parameters and you want to specifically remove the tag with those parameter values.
89. **RemoveTextOverlay** – this removes any text overlay present on the screen. There are no parameters.
    - Note: Text overlays are removed automatically at the end of a player's turn. This action is only for instances where you want the text overlay to go away in the middle of a turn.
90. **RemoveTrigger** – Takes two parameters: Y Coordinate and X Coordinate. Removes all triggers from all objects found at those coordinates. (*See* Triggers within Section G, Maps, below.)
91. **RotateChar** – this rotates the named character to face a direction of your choice. There are two parameters: the name of the character to rotate, including spaces; and the direction to face. The following are available directions you can use:
    - Up
    - Down
    - Left
    - Right

    - ♦ Usage example: **RotateChar Jim Francis,Right** will cause the character named Jim Francis to face right.

92. **RotateCharToFace** – this rotates the named character to face another character of your choice. There are two parameters: the name of the character to rotate, including spaces; and the character to face.
    - This action is useful for making characters talk to one another when you cannot be certain of their positions ahead of time (e.g. because the conversation doesn't occur on turn 0). It is recommended that you use RotateCharToFace instead of RotateChar in most situations where characters face toward one another.
        - ♦ Usage example: **RotateChar Jim Francis,Claude Mann** will cause the character named Jim Francis to face Claude Mann, regardless of where each character is on the battlefield.
93. **RotateCharTowardPoint** – this rotates the named character to face a spot on the battlefield. There are three parameters: the name of the character to rotate, including spaces; the Y Coordinate of the spot to face; and the X Coordinate of the spot to face.
94. **Run** – runs a script. Run takes only one parameter: the name of the script to run.
95. **ScoreBonus** – affect a player's bonus points for purposes of determining the player's score at the end of the battle. There are two parameters to the ScoreBonus action: Army Number and Amount. The Amount will be added directly to that player's bonus points. To penalize a player's score, simply use a negative number for the Amount.
96. **SetArr –** alter the contents of an array entry if it exists; add the entry if it doesn't; and also create the array if the named array doesn't yet exist. There are three parameters to the SetArr action: Array Name (a string with the name of the array you want to edit an entry in/add an entry to); Add at Position (*front*, *end*, or a specific integer reflecting the position of an existing entry to overwrite); and the string or value to use for the entry.
97. **SetStat** – alter the value of a character's stat. There are four parameters to the SetStat action: Character Name with spaces, Stat, Operation, and Amount.
    - The amount must be an integer; the operation to be performed with the amount can be any of the following:
        - = (make variable equal to amount)
        - + (add amount to variable)
        - - (subtract amount from variable, with results constrained to 0 or higher)
        - -- (subtract amount from variable, with negative results allowed)
        - * (multiply variable by amount)
        - / (divide variable by amount; the result will be rounded to the nearest integer value)
        - % (multiply variable by amount as a percentage; the result will be rounded to the nearest integer value)
    - The Stat can be any of the stat types supported by **IfStatGoTo** above *except* for Health and Energy—use Damage and Drain instead to affect a character's current health and energy stats.
    - In addition, SetStat supports the *Name* stat.
    - To edit string stats (*Name*, *Class*, *Sprite, Portrait*, *Sex*, *Race*, *Direction*, or

MoveType), add a fifth parameter with the replacement name (delimited by a colon), the replacement portrait name (minus the .png file extension), the replacement class name, the replacement sprite name, the replacement sex, the replacement direction (*Up*, *Down*, *Left*, *Right* or *None*), or the replacement movement type (*flying* or *land*). The operation and amount parameters must still be present, but it doesn't matter what they say in this instance.

- In addition to changing ordinary character stats, **SetStat** can also be used to change a character's leveling behavior.
  - ♦ To change the likelihood of a character gaining points in a certain stat on level up, use the normal stat name with *LVL* and a space in front. (Instead of *Strength*, for instance, you'd use *LVL Strength*.)
    - ▫ This can be used to affect any of the stats normally found in a character's *OnLevelUp* tag within *CharClasses.xml*.
  - ♦ To queue up a new attack for a character to learn at a certain level, use *AttackToLearn* as the Stat parameter, the attack's name for the Operation parameter, and the level at which the attack is to be learned for the Amount parameter.
- Note that any modifications you make with SetStat are permanent, and will affect the character going forward in the campaign. (This does not apply to Damage, Drain, Done, Attacked, Countered, or Steps Left.)
  - ♦ <u>Usage example</u>: **SetStat Jim Francis,Strength,%,150** will look for the character named Jim Francis; if he is found on the battlefield, the game will permanently increase his Strength stat to 150% of its current value.
  - ♦ <u>Usage example</u>: **SetStat Jim Francis,Name,+,0,General:Francis** will look for the character named Jim Francis; if he is found on the battlefield, the game will permanently change his name to "General Francis" (with the colon separating his first and last name).
  - ♦ <u>Usage example</u>: **SetStat Jim Francis,LVL Max Health,+,1** will look for the character named Jim Francis; if he is found on the battlefield, the game will permanently increase the chance that his maximum health increases upon leveling up.
  - ♦ <u>Usage example</u>: **SetStat Jim Francis,AttackToLearn,Mind Blast,8** will look for the character named Jim Francis; if he is found on the battlefield, the game will make it so he learns Mind Blast upon reaching level 8.

98. **SetStatByStat** – alter the value of a character's stat by reference to another stat (either from the same character, or from a second character). There are five parameters to the SetStatByStat action: Character Name with spaces, Stat, Operation, Second Character Name with spaces, and Second Stat.
    - <u>Note</u>: this only works with numerical stats; string stats like MoveType or Class will not work!
      - ♦ <u>Usage example</u>: **SetStatByStat Jim Francis,Strength,=,Emma Strider,Strength** will look for the character named Jim Francis; if he is found on the battlefield, the game will permanently set his Strength stat equal to Emma

Strider's Strength stat.

99. **SetStatByVal –** alter the value of a character's stat by reference to the value of a custom variable. There are four parameters to the SetStatByVal action: Character Name with spaces, Stat, Operation, and Custom Variable name.
   - Note: this only works with numerical stats; string stats like MoveType or Class will not work!
     - ♦ **Usage example: SetStatByVal Jim Francis,Perception,+,OculoFavor** will look for the character named Jim Francis; if he is found on the battlefield, the game will permanently increase his Perception stat equal to the custom variable OculoFavor (which might, in turn, be a variable that tracks how much Oculo the lensmaker likes the player).

100. **SetString** – alter the value of a string custom variable if it exists, or create the variable if it doesn't. There are two parameters to the SetString action: Variable Name and String. The String can be any word or phrase you wish.
     - ♦ **Usage example: SetString AlliedWith,Imperials** will set the custom string variable *AlliedWith* to *Imperials*, perhaps to indicate that the player has chosen to ally with the imperial faction in the game world. (If the *AlliedWith* variable didn't already exist, this action creates it as well.)

101. **SetStringByString –** copy the string held in one custom string variable into another. There are two parameters to the SetStringByString action: Custom Variable Name to paste the string to, and Custom Variable name to copy the string from.

102. **SetVal** – alter the value of a numerical custom variable if it exists, or create the variable if it doesn't. There are three parameters to the SetVal action: Variable Name, Operation, and Amount. The amount must be an integer; the operation to be performed with the amount can be any of the following:
   - = (make variable equal to amount)
   - + (add amount to variable)
   - - (subtract amount from variable, with results constrained to 0 or higher)
   - -- (subtract amount from variable, with negative results allowed)
   - * (multiply variable by amount)
   - / (divide variable by amount; the result will be rounded to the nearest integer value)
   - % (multiply variable by amount as a percentage; the result will be rounded to the nearest integer value)
   - r (randomly select a positive integer between 1 and the named amount)
     - ♦ Usage example: **SetVal Money,+,100** will add 100 to the custom variable Money; if there isn't any existing custom variable named Money, however, the game will create that custom variable with a value of 0, then proceed with the operation (in this case, setting its value to 100).

103. **SetValByStat –** alter the value of a custom variable by reference to a character stat. There are four parameters to the SetValByStat action: Custom Variable name, Operation, Character Name with spaces, and Stat.
     - ♦ **Usage example: SetValByStat IntimidationLevel,+,Emma Strider,Strength** will look for the character named Emma Strider; if she is found

on the battlefield, the game will add Emma Strider's Strength stat to the custom variable IntimidationLevel.

104. **SetValByVal** – alter the value of a custom variable by reference to the value of a second custom variable. There are three parameters to the SetValByVal action: Custom Variable name, Operation, and Second Custom Variable name.
    - ♦ <u>Usage example</u>: **SetValByVal MerchantReputation,*,Money** will multiply the custom variable MerchantReputation by the value of the custom variable Money. (If the player has 100 money, for instance, this would multiply MerchantReputation by 100.)
105. **ShakeScreen** – this shakes the screen when the conversation reaches the current branch of dialog. There are two parameters: the magnitude of the shaking and the length of the shaking. Magnitude is a positive integer referring to the maximum number of pixels the screen can be displaced with each frame of shaking; length is the number of frames the screen will shake for.
106. **ShowActionsMenu** – no parameters. This causes the actions menu to appear onscreen even though the conversation hasn't ended. This is primarily for use in tutorials.
107. **ShuffleArr** – randomize the order of the entries in a custom array. There is one parameter to the ShuffleArr action: Array Name (a string with the name of the array you want to randomize the entries within).
108. **SpawnChar** – this spawns a new character on the battlefield. There are five parameters: the team number; the name of the character to spawn, with a colon delimiting the first and last name; the Y coordinate to spawn on; the X coordinate to spawn on; and the direction to face upon spawning.
    - You may use an optional sixth parameter, Trigger: this functions just like triggers that are added directly to a <Unit> tag within an individual map (*see* ***Section G*** below), except that the trigger type and script name are delimited with a colon instead of a comma.
    - You may also use an optional seventh parameter, Tags: this functions just like tags that are added directly to a <Unit> tag within an individual map (*see* ***Section G*** below), except that the Add/Remove, Tag Name and Tag Parameters of each tag are delimited with colons instead of commas, and individual tags are delimited with double-colons (**::**).
    - The team number for the player is always 0; the enemy's team number is always 1; destructible objects always belong to team 99. Use these numbers for the first parameter accordingly.
    - The reason that the name is delimited here rather than typed out with spaces is because the game is going to look directly into the *CharClasses.xml* file and compare what you put here with the *charname* attribute of every character. Make sure you use a colon and not a space in the second parameter!
    - If you're spawning a destructible object, use its *spritetype* for the first parameter instead—no colon and no spaces. Also, make sure to set its facing to *None*!
        - ♦ <u>Usage example</u>: **SpawnChar 0,Jennifer:Faust,11,5,Up** will spawn the character named Jennifer Faust on the player's team (team 0), facing up, on the space 11 rows down from the top and 5 columns right from the left side of the

battlefield.

- ♦ Usage example: **SpawnChar 1,Bandit:Swordsman,3,5,Down,, Add:Passive::Add:LevelUp:2** will spawn a bandit swordsman on the enemy's team (team 1), facing down, on the space 3 rows down from the top and 5 columns right from the left side of the battlefield. It will then add Passive and LevelUp tags to the character.
- ♦ Usage example: **SpawnChar 99,PressurePlate,3,5,None,Pressure:Trap** will spawn a pressure plate on the space 3 rows down from the top and 5 columns right from the left side of the battlefield. Because it is a destructible object, it does not “face” in any direction so far as the battle engine is concerned. The optional sixth parameter gives it a Pressure trigger that will run the script named *Trap* as soon as a character steps on it.

109. **SpawnFloatingText** – adds a short-lived animated text pop-up above a specified character. There are four parameters: the text to use, the name of the character to add it above, the color of the text in hexidecimal format, and the number of frames to delay the text. (The third and fourth parameters are optional; if not added, the game will assume that the text is yellow and that there is no delay.)
    - Note: do not use commas in the text defined by the first parameter or the game will glitch out.
110. **SpawnFloatingTextAt** – adds a short-lived animated text pop-up above a specified character. There are five parameters: the text to use, the Y coordinate, the X coordinate, the color of the text in hexidecimal format, and the number of frames to delay the text. (The fourth and fifth parameters are optional; if not added, the game will assume that the text is yellow and that there is no delay.)
    - Note: do not use commas in the text defined by the first parameter or the game will glitch out.
111. **SpawnParticlesAt** – adds a short-lived spray of particles to the battlefield at a location you specify. There are three parameters: the Type of Particle to use, the Y coordinate, and the X coordinate. Particles you can use for the first parameter include:
    - *Sparks*
    - *Blood*
    - *Wood*
    - *Stone*
    - *Water*
    - *Spray*
    - *Smoke*
    - *Snow*
112. **SpawnRandomly** – this is the same as SpawnChar, except that the location where the character or object spawns is chosen randomly by the game. The game will pick a space that is passable by non-flying characters, unoccupied by other characters, and free of any non-bridge destructible objects. The parameters are the same as with SpawnChar, but minus the X and Y coordinate parameters: team number; the name of the character to spawn, with a colon delimiting the first and last name; the direction to face upon

spawning; trigger (optional); and tags (optional).

- Note: Unlike the ItemDrop action, SpawnRandomly does *not* enforce any sort of minimum spawn distance away from other characters, so it is possible for characters or objects to spawn right next to the player!

113. **TeachAttack** – teach a character a new attack. The character gains that attack permanently. There are two parameters: the Character's Name, including spaces; and the Name of the Attack to gain.
    - Note: this action can be used in a Static Cut Scene, but it will work only on characters who have already appeared in at least one battle on the side of army 0. You must delimit the character's first and last names with a colon if you use the TeachAttack action in this context.
114. **TeachAttackTemp** – teach a character a new attack. The character gains that attack only until the end of the battle. There are two parameters: the Character's Name, including spaces; and the Name of the Attack to gain.
115. **TransferChar** – this removes a character from any existing army rosters where it may appear, then *if the character was found and removed successfully,* it then adds that character to a specific army roster. (In essence, this lets you switch characters between armies while keeping characters who have suffered permadeath dead.) There are two parameters: the Name of the Character to transfer, with a colon delimiting the first and last name; and the Roster Number for the army to transfer that character to if the character is found alive.
    - ♦ Usage example: **TransferChar   Jennifer:Faust,1** will cause the game to remove the character named Jennifer Faust from any existing army rosters in which she appears, then—if the game found her alive in an army roster—add her to army roster 1.
116. **Unlock** – this looks for a door at a specified set of battlefield coordinates; if it finds a door at those coordinates, it unlocks and opens it. There are two parameters: the Y coordinate, and the X coordinate.
117. **WhoCanUse** – Gathers the names of all characters on the battlefield who can use a particular item, then automatically creates and places their names into a custom string named *_WhoCanUse*. One parameter: item name.

- **Types of Replies**
- Replies are used to control the flow of conversation, and can present the player with options for responding to character dialog and prompts from the game. Note: if a branch has only a single reply option, the reply will not be displayed; the game *only* displays the replies individually where the player has a choice of responses (i.e. two or more available replies).

1. **NextBranch –** advances the conversation to the next consecutive branch (to branch 1 if on branch 0, to branch 5 if on branch 4, and so on). No parameters.
2. **NewBranch** – takes one parameter: branch number. Advances the conversation to the branch specified in the parameter.
    - ♦ Usage example: **Nope.   NewBranch   1** will move the conversation to branch 1. If there is more than one reply, it will display the text “Nope.”
3. **EndConv** – no parameter. Ends the conversation and returns to player control of the battle.

- ♦ Usage example: **Let's talk later.** **EndConv** will end the conversation.

4. **LastBranch** – returns to a branch in the conversation that the player was on before the current branch. By default, this just returns to the branch the player was on immediately prior to the current branch. There is an optional parameter you can use with LastBranch, however: number of branches to move back in the conversation.
   - ♦ Usage example: **Sorry, I'll ask a different question.** **LastBranch** will move the conversation back to the previous branch. If, for instance, the player got to the current branch by clicking a reply in branch 3, then clicking this reply will return to branch 3.
   - ♦ Usage example: W**e've made a mistake; let's try again.** **LastBranch 4** will move the conversation back four steps. If, for instance, the player got to the current branch by clicking a reply in branch 13, which he reached by clicking a reply in branch 12, which he reached by clicking a reply in branch 8, which he reached by clicking a reply in branch 5, then clicking this reply will return to branch 5, four steps back (current branch > 13 > 12 > 8 > 5).
5. **LastLastBranch** – takes no parameters. Simply returns to two branches back in the conversation.
6. **LastLastLastBranch –** takes no parameters. Simply returns to three branches back in the conversation.
7. **NamedBranch –** takes one parameter: branch name. The game goes to the dialog branch with a name attribute that matches it.
8. **CheckForMoreDialog** – no parameter. Checks to see if any other conversations are queued up. If so, it moves straight to the next conversation; if not, it ends the conversation a la EndConv. The same as the CheckForMoreDialog action, in essence, but it activates upon clicking a reply rather than upon reaching a new branch.
9. **EndTurn** – no parameter. Ends the conversation, as well as the current player's turn.
10. **EndBattle** – no parameter. Ends the battle entirely, bringing up the normal splash screen indicating that the player won.
11. **LoseBattle** – no parameter. Ends the battle entirely bringing up the normal splash screen indicating that the player was defeated.
12. **NextScene** – no parameter. Ends the conversation and immediately loads whatever scene is specified in the *nextbattle* attribute of the current map.
13. **NewScene** – takes one parameter: the name of the map to load next. Ends the conversation and immediately loads the map specified in the parameter.
    - ♦ Usage example: **Then it's settled: we march north.** **NewScene NorthernMarch** will end the conversation and begin loading the map *NorthernMarch.xml*. If there is more than one reply, it will display the text "Then it's settled: we march north."

E. Scripts

- ○ If you're familiar with programming, a Script is basically a function; it is a bundle of dialog actions grouped together under a single name. A script can be run during dialog simply by referencing the name in a **Run** action. This makes it easy to create complicated effects that can be referenced over and over again.
- ○ Scripts can be created within an individual map, or in *PersistentDialog.xml*.

- Scripts use the <Script> tag, and contain <Action> tags just like dialog does.
- Instead of having a whole bunch of free-floating data lying around within the <Script> tags identifying a speaker, dialog text, and so on, however, there is only one bit of text: the name of the script. This is how this looks in practice:
  - ♦ Usage example: **<Script>Nightfall**
    <Action>ChangeCondition/Global Lighting,Night</Action>
    <Action>ChangeCondition/Fog of War,true</Action>
    </Script>
    creates a script named Nightfall that will impose dark global lighting on the level and turn on fog of war.

F. Items

- To mod items, open up *ItemClasses.xml* in a text editor of your choice.
- Each item has the following properties, in order:
  1. **name** – the item's name.
  2. **useableWith** – this tells the game whether an item must be triggered, equipped, or whether it takes effect automatically upon pickup. You can use "automatic" to make the item trigger immediately on pickup, "triggered" to let the player use it manually, or you can name one of the game's eight equipment slots if you want the item to be equippable (in which case it will become equippable only in that slot). Alternatively, if you want to make an item undroppable and unusable, select make it useableWith "quest":
     - *automatic*
     - *triggered*
     - *Weapon Hand*
     - *Off Hand*
     - *Head*
     - *Neck*
     - *Torso*
     - *Back*
     - *Feet*
     - *Accessory*
     - *quest*
  3. **requirements** – this tells the game what characteristics a character has to possess before he or she can either use or equip an item. There are three requirement types: race, class, and level. An item may use any combination of these three.
     - *Race requirements* – To create race requirements, type *r*, a colon, then the names of every race that can use or equip the item, separated by commas. This will tell the game that any of the named races (and only those races) may use the item.
     - *Class requirements* – To create class requirements, type *c*, a colon, then the names of every class that can use or equip the item, separated by commas. This will tell the game that any of the named classes (and only those classes) may use the item.
     - *Level requirement* – To create a level requirement, type *l*, a colon, then the minimum required level to use or equip the item.
       - ♦ Usage example:

requirements="r:Human,Spriggat/c:Swordsman,Fencer/l:6" will tell the game that the item can only be used or equipped by a character that is a Human or a Spriggat, that is either a Swordsman or Fencer, and that is level 6 or higher.

4. **endsStatus** – type in the name of a status effect you want the item to end. To end multiple status effects, type in each status effect's name separated by a forward slash.
   - NOTE: you can type *all* in lieu of specific status effects if you want the item to remove every status effect on the character.
5. **addsStatus** – type in the name of a status effect you want the item to grant. To grant multiple status effects, type in each status effect's name separated by a forward slash.
6. **grantsAtk** – type in the name of an attack you want the item to grant. To grant multiple attacks, type in each attack's name separated by a forward slash.
   - NOTE: attacks granted by triggered and automatic items will last for the remainder of the battle. Attacks granted by equipped items will remain for as long as the item is equipped.
7. **consumedAfter** – for *triggered* and *automatic* items, the maximum number of times an item can be used before it will be fully consumed and vanish. For items *useableWith* Weapon Hand, if this is set to any number above 0, that is the number of attacks the weapon will last for. (Attacking with a weapon that has 1 use left will cause the weapon to break.)
   - To have weapons last forever, just leave *consumedAfter* set to 0.
8. **itemValue** – can be grabbed to set a custom value in the game using the **GetItemValue** script action.
9. **hpPlus** – increase character's current health. (Negative numbers will damage character health.)
10. **pspPlus** – increase character's current energy. (Negative numbers will damage character energy.)
11. **maxHPPlus** – increase character's maximum health. (Negative numbers will damage maximum character health.)
12. **maxPsPPlus** – increase character's maximum energy. (Negative numbers will damage maximum character energy.)
13. **spdPlus** – increase character's current speed. (Negative numbers will lower character speed.)
14. **dodgePlus** – increase character's current dodge percentage. (Negative numbers will lower character dodge.)
15. **strPlus** – increase character's current strength. (Negative numbers will lower strength.)
16. **perPlus** – increase character's current perception. (Negative numbers will lower perception.)
17. **psyPPlus** – increase character's current psy power. (Negative numbers will lower psy power.)
18. **psyDPlus** – increase character's current psy defense. (Negative numbers will lower psy defense.)
19. **prcResPlus** – increase character's current pierce resistance. (Negative numbers will lower pierce resistance.)
20. **slshResPlus** – increase character's current slash resistance. (Negative numbers will

lower slash resistance.)

21. **crshResPlus** – increase character's current crush resistance. (Negative numbers will lower crush resistance.)
22. **mnResPlus** – increase character's current mental resistance. (Negative numbers will lower mental resistance.)
23. **htResPlus** – increase character's current heat resistance. (Negative numbers will lower heat resistance.)
24. **cdResPlus** – increase character's current cold resistance. (Negative numbers will lower cold resistance.)
25. **ltResPlus** – increase character's current light resistance. (Negative numbers will lower light resistance.)
26. **shResPlus** – increase character's current shadow resistance. (Negative numbers will lower shadow resistance.)
27. **poiResPlus** – increase character's current poison resistance. (Negative numbers will lower poison resistance.)
28. **accPlus** – increase character's current base attack accuracy percentage. (Negative numbers will decrease character attack accuracy.)
29. **ctrLimitPlus** – increase character's current counter limit. (Negative numbers will decrease the character's counter limit.)
30. **commonality** – type a positive integer representing the item's commonality. The higher the number, the more common it is.
    - NOTE: in multiplayer, this directly impacts the probability of an item dropping in a random item drop. In both multiplayer and single player, it also affects the likelihood of an item appearing in the inventory of a character or object in place of an *-R-* symbol; the higher the commonality value, the more likely it is to be randomly selected. Similarly, it determines whether the item falls within the commonality range of a targeted *R[x-y]* symbol. (For example: a chest with *R[12-15]* in its inventory will spawn with a randomly selected item that has a commonality value between 12 and 15.)
31. **addsTags** – tags to add to the character upon using or equipping the item. The game will remove these tags from the character again upon unequipping the item (assuming that it was an equipped item). Tags consist of the tag name; if the tag has any attributes, place a comma between the tag name and its attributes, delimiting each attribute with a colon. To add multiple tags, delimit each with a forward slash. Tags are discussed in more detail in **B. Characters and Destructible Objects** below.
32. **image** – the filename of the item's icon, minus the *.png* file extension. This is used to display the item graphically in the player's inventory. If the game can't find the named image, it will display the name of the item instead.
33. **description** – the text description of the item as it appears in-game.

○ In addition to all of the above, you can have items **run a script** by placing the name of the script in between the Item tags. (The script, in turn, should probably be placed in *PersistentDialog.xml* if you want it to work in all maps!)

♦ Usage example: **<Item … >Gain25Coins</Item>** tells the game to run the script named Gain25Coins as soon as the item is used. If the item's **useableWith** attribute is set to *automatic*, this will instead happen as soon as the item is picked up.

G. <u>Maps</u>

- To **edit maps**, use the map editor.
- To **add conditions** to a map, see **C. <u>Conditions</u>** above.
- To **add dialog and scripts** to a map, see **D. <u>Dialog</u>** above.
- To **add an objective** to a map, open up the map in a text editor of your choice and create a new <Objective></Objective> tag; then type the text you want the game to use within it.
    - ♦ <u>Usage example</u>: **<Objective>Protect the gates!</Objective>** will cause the text "Protect the Gates!" to show up in the Objectives window for this map.
- To **add character tags** to a character or destructible object within a map, open up the map in a text editor of your choice. Within the chosen unit's <Unit> tag, add a *tag* attribute followed by *Add* or *Remove*, a comma, and the tag (and any associated parameters) you wish to add. To add or remove multiple tags, delimit each with a forward slash. Tags are discussed in more detail in **<u>B. Characters and Destructible Objects</u>** above.
    - <u>Note</u>: you *must* include *Add,* or *Remove,* before each character tag if you stick them directly within <Unit>! You do not need to do this for character tags that are added via *CharClasses.xml* or *ObjClasses.xml*.
- To **add personalized custom lighting** to a particular character or destructible object within a map, open up the map in a text editor of your choice. Within the chosen unit's <Unit> tag, add a *lighting* attribute with three parameters, each delimited by commas: Red, Green and Blue values. These are just like the parameters in the *lighting* attribute assigned to characters in *CharClasses.xml* and *ObjClasses.xml*: they are decimal numbers that can be anywhere from 0 to 2.0, with 1.0 being 100% color value (0,0,0 will make the character pitch black; 1.6,1.2,0.7 will make the character glow orange; and so on).
    - <u>Note</u>: unlike the *lighting* attribute in *CharClasses.xml* and *ObjClasses.xml*, this only affects the particular instance of the unit you assign it to, and it only lasts until the end of the scene or battle.
- To **add a trigger** to a destructible object within a map, open up the map in a text editor of your choice. Within the chosen unit's <Unit> tag, add a *trigger* attribute with two parameters delimited by a comma: trigger type and the name of the script to trigger. There are three acceptable trigger types:
    1. *Pressure* – activates as soon as a character steps on it. Can be activated again later.
    2. *Use* – activates as soon as a character uses it. Can be activated again later.
    3. *Use Once* – activates as soon as a character uses it. Can only activate once.
        - ♦ <u>Usage example</u>: **<Unit trigger="Use,UseSwitch">0,99,Switch,0,6,None,None</Unit>** tells the game to stick a *Use* trigger on a switch. It will run the script named *UseSwitch* when activated. The switch can be activated an unlimited number of times.
- To **add lighting** to a map, open up the map in a text editor of your choice. Add a <Light></Light> tag with the following properties in the middle.
    1. *Y coordinate*
    2. *X coordinate*

3. *Red value* (whole number, 0-255)
4. *Green value* (whole number, 0-255)
5. *Blue value* (whole number, 0-255)
6. *Diameter* (in number of pixels)
7. *Intensity* (any number between 0 and 1)
8. *Flicker* (currently does nothing; leave at 0)
    - ♦ Usage example: **<Light>0,5,120,220,255,80,0.3,0</Light>** tells the game to load the map with a light centered in the top row of the map, five tiles from the left; that the light should be turquoise in color (Red: 120 Green: 220 Blue: 255); and that it should be 80 pixels in diameter, rendered at 30% intensity.

- ○ To **add music to a map**, just change what's written in the map's *musictrack* attribute. Available music includes the following tracks:
    1. *BadOmen* – forboding music anticipating danger.
    2. *BattlePrep* – an upbeat military march.
    3. *CheckingOutTheGoods* – bossa nova music for romance and casual shopping.
    4. *CityOfTheEmpire* – music for a bustling city.
    5. *DeviousSchemes* – high-stakes stealth music.
    6. *Dungeon* – creepy music for exploring a dungeon.
    7. *EvilLurks* – skin-crawlingly creepy music for deeply evil characters.
    8. *FinalBoss* – epic battle music for the final boss.
    9. *ForHonor* – battle music; the heroes are fighting for something greater!
    10. *FoulServants* – boss music.
    11. *FrozenMemories* – sad music; for defeat, or at touching or nostalgic moments.
    12. *HerosTriumph* – joyous music for a great victory!
    13. *LizardTribe* – jazzy tribal music with a digeridoo.
    14. *MythicalCreatures* – mysterious music with tribal percussion.
    15. *Onslaught* – battle music; standard battle theme.
    16. *Resolve* – music indicating steely resolve in the face of adversity.
    17. *RightUnderTheirNoses* – light-hearted stealth music.
    18. *ScionOfEvil* – boss music.
    19. *ShadyTransactions* – music for talking with mercenaries and arms dealers.
    20. *Sisters* – sweet music; for times the characters reveal their feelings.
    21. *TacticiansDuel* – battle music; alternate battle theme.
    22. *TavernTheme* – an upbeat folk tune.
    23. *ThereWillBeBlood* – forboding battle music; the calm before the storm.
    24. *TitleTheme* – the Telepath Tactics theme song.
    25. *VibraMines* – depressing, desolate music for the oppressed.
    26. *VillageTheme* – gentle, peaceful music for a quiet village.
- ○ To tell the game **what scene to go to next** after the battle is won, just change what's written in the map's *nextbattle* attribute. Use the filename of the battle or cut scene you want to load, minus the ".xml" extension.
    - Note: to **generate a random level**, use *GENERATE_RANDOM_LEVEL[]*. Within the square brackets, delimited by forward slashes, include the following parameters:
        1. *dungeon type* – tells the game which rooms to use when building the level;

specifically, which subfolder within *Maps* > *Generator Chunks* to use. (If, for instance, you've created a folder within Generator Chunks called Forest with a bunch of rooms that are just grass and trees and bushes and such, and you want the game to generate a level using those chunks, you'd use *Forest* for this parameter.)

2. *nextbattle* – the name of the scene to go to once the level is successfully completed (see above).
3. *musictrack* – the music to play for this level (see above).
4. *number of floors* – how many levels the generated dungeon should contain. (Leave this at 1 for now.)
5. *level size* – a number representing both the width and height of each level; room chunks are each 7x7, and so this parameter must be a multiple of 7 (e.g. *21* will produce 21x21 floors with space for up to 3 rooms on a side).
6. *number of enemies* – a range of enemies, with the lower bound and upper bound delimited by a hyphen (e.g. *8-12* will cause the level to spawn between 8 and 12 enemies).
7. *level of enemies* – a range of enemy levels, with the lower bound and upper bound delimited by a hyphen (e.g. *1-3* will result in all enemies spawned being between level 1 and level 3).
8. *types of enemies* – the names of all possible enemies that can spawn, with a comma between first and last names, with each individual enemy delimited by a colon (e.g. *Bloodbeard's,Bandit:Bloodbeard's,Bowman: Bloodbeard's,Healer* will tell the game to choose from these three types of enemies when populating the level).
9. *conditions* – all Conditions you want the level to have, from global lighting to deployment. Delimit each condition's parameters using commas, as usual; the conditions should each be separated by a colon (e.g. *Global Lighting,Cave:Protect Char,0,Lorenzo Llamas* will set the level's lighting and make it so the player has to keep the character Lorenzo Llamas alive).

♦ Usage example: **GENERATE_RANDOM_LEVEL[Ancient Dungeon/Exit Cut Scene/Dungeon/1/30/10-14/2-6/Ancient,Ghost:Ruins,Bandit,: Ruins,Swordsman/Global Lighting,Cave]** will cause the game to generate a single 30 x 30-tile level using room chunks from the "Ancient Dungeon" subfolder, lit with the "Cave" global lighting condition, playing the "Dungeon" music track, containing between 10 and 14 enemies (a mix of Ancient Ghosts, Ruins Bandits and Ruins Swordsmen) ranging in level from 2 to 6, and proceeding to the scene *Exit Cut Scene.xml* upon completion.

H. Cut Scenes

- Cut scenes come in two varieties: Scrolling and Static.
  1. **Scrolling cut scenes** feature text that scrolls from the bottom of the screen up to the top across a black background. The scene ends when the text vanishes off the top of the screen.
  2. **Static cut scenes** are built in a slideshow format; the player clicks through a succession of text and images. Static cut scenes also support character dialog.

- To create a cut scene, take a Map file and change the *maptype* attribute:
  - to create a scrolling cut scene, use *maptype="Cut Scene: Scrolling"*
  - to create a static cut scene, use *maptype="Cut Scene: Static"*
- Cut scenes use a few unique tags:
  1. **<Narration>** – this is the central tag used in a cut scene; this defines the text that is going to scroll up the screen (or appear in slides, in the case of a static cut scene).
     - Within a static cut scene, divide the narration text into individual slides using a double-forward slash. This, for instance, will create two slides:
       - Usage example: **This will be slide 1.//This is slide 2, and will appear after the player clicks.**
     - You can embed a background picture in a static cut scene by including text with this format: *-PIC:Name of the picture-*
       - Usage example: **-PIC:Title Screen-** will make the game load *Data > Backgrounds > Title Screen.png* as a background image starting with the slide where this was placed.
     - You can remove the background picture (i.e. make the area above the text go black) by using *-PIC:None-* or *-PIC:Clear-* instead.
     - You can also embed dialog in a slide within a static cut scene using this format: *-DIA:TriggerType/TriggerParameters-*
       Note that dialog will obscure any narrative text. Further, ending the conversation will automatically advance the static cut scene to the next chunk. Thus, it is best to avoid using narrative text in the same chunk as cut scene dialog.
       - Usage example: **-DIA:OnTurn/0-** will cause the dialog with the OnTurn/0 trigger to begin.
  2. **<NewRoster>** – include character names in forward-slash notation, delimited by commas, to create the player's starting roster at the beginning of the game. If you use this tag later in the game, it will replace the player's current roster with the named characters.
       - Usage example: **<NewRoster>Gambino/Pelosi,James/Francis</NewRoster>** will create a new army roster consisting of the characters Gambino Pelosi and James Francis.
  3. **<AddToRoster>** – include character names in forward-slash notation, delimited by commas, to add those characters to the player's current army roster.
       - Usage example: **<NewRoster>Jennifer/Faust</NewRoster>** will add the character Jennifer Faust to the player's army roster.
  4. **<RemoveFromRoster>** – this works exactly like **<AddToRoster>**, only it removes the named characters rather than adding them.

I. <u>Tilesets</u>

- To create a new tileset, go to the subdirectory *Data > Tiles*.
- Next, create a new folder within *Tiles*; name the folder whatever you want the name of your new tileset to be.
- Create your tiles. You can use just about any paint program for this purpose; just make sure the tiles are 64 x 64 .pngs.

- Drop your tiles into the folder you just created.
  1. Tiles must *all* be 64 x 64 pixel .png files.
  2. All tiles must be sequentially numbered in the following manner: *tile0001.png*, *tile0002.png*, *tile0003.png*, etc.
- Once you have done this, open the map editor. It should auto-detect your new tileset!
- To make the tileset work in Telepath Tactics itself, however, you must do one more thing: create a *TileData.xml* file for your tileset.
- First, copy-paste *TileData.xml* from the Blank tileset into your new tileset directory. Now open it in a text editor of your choice.
- At the top you will see a line that looks like this:
  <TileData setDir="app:/Tiles/Blank"> Replace the word Blank with the name of your tileset. Make sure it matches the folder name exactly; if it doesn't, the game will fail to load the properties of your tiles, which will lead to other errors in turn!
- Next, you should see a line that looks like this:
  <Tile fileName="tile0001.png" passability="none" dmg="0" element=""></Tile>
  You need to create one of these lines for every tile in your tileset. Each of these lines has the following attributes:
  1. **fileName** – this is the name of the .png file whose properties you're setting. Make sure you do them in order!
  2. **passability** – this determines which characters can move through a tile. Possible settings are:
     - *none* – this tile is treated like the inside of a solid wall.
     - *flying* – only flying characters can pass (unless a bridge is placed on the tile).
     - *all* – any character can pass.
  3. **dmg** – this tells the game how much damage a non-flying character takes for beginning its turn on this space. (For most spaces, this is 0.)
  4. **element** – this tells the game the element of the damage the character takes. If a word is used that doesn't represent an in-game element (e.g. Water) is used, the game will ignore all resistances and deal the full amount of the damage to the character. If the space deals no damage, leave this blank.
- In addition, any tile can have an additional attribute, right after **element**, called **special**. This gives the tile a terrain effect that can impose one or more stat bonuses or penalties to any character standing on it. For each stat to effect, use two values delimited by a comma: the stat name and the amount that gets added to the stat (this can be a negative value if you want it to be a penalty instead of a bonus). To affect multiple stats, delimit each with a forward slash.
  - **special="Accuracy,25/Dodge,10/Resistance,10"** , for example, would give whatever character stands on that tile bonuses of 25 points to accuracy, 10 points to dodge, and 10 points to each type of resistance for as long as the character remains on that tile.
  - For a list of the stats this can affect, see the **Space Bonus** condition above.
- Once you've finished filling out *TileData.xml*, your tileset will be good to go. Telepath Tactics will automatically use it whenever it detects a map that uses the tileset.

- Note: if you want to distribute a map that uses a custom tileset, you *have* to distribute the tileset as well, or the map won't load! To distribute your custom tileset, just stick the folder in a .zip file and have other players unzip the tileset folder into *Data > Tiles*.

J. <u>Destructible Object Sprites</u>

- To create a new destructible object sprite, open up an image editor and create a new image 64 pixels wide with a transparent background.
- Draw the object sprite.
- Save the sprite in the subdirectory *Data > Objects* as a .png file.
- To use the sprite, set the **spritetype** property of a destructible object within *ObjClasses.xml* to the filename you used, minus the .png file extension. (For instance: if you just created a wooden carriage object sprite called *Carriage.png*, you'd use the **spritetype** *Carriage* to tie it to a destructible object class.)

K. <u>Shadow sprites</u>

- This is exactly like creating a destructible object sprite, with one difference: the filename must begin with *Shadow.*
- To use the shadow, set the **shadowtype** property of a character or destructible object to the filename you used, minus the "Shadow" prefix and minus the .png file extension. (For instance: if you just created a shadow sprite called *ShadowDiamond.png*, you'd use the **shadowtype** *Diamond* to use the shadow with a character or object class.)

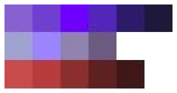
L. <u>Character Sprites</u>

- Characters use sprite sheets to animate. There are two basic kinds of character animations: attack animations and non-attack animations. Attack animations have larger frames and appear in a separate sub-directory, but otherwise behave the same way as non-attack animations.
- <u>Tools</u>
  - For creating sprite sheets, I strongly recommend <u>Pyxel Edit</u> or <u>Graphics Gale</u>. (You could also use a free online editor like <u>Piskel</u> in a pinch.)
- <u>File type</u>
  - As with destructible objects, you must use *.png* files with transparent backgrounds.
- <u>Orientation</u>
  - Every character animation is organized into four rows, each representing a different direction the character might be facing. From top to bottom, these are:
    1. *Down-facing*
    2. *Left-facing*
    3. *Up-facing*
    4. *Right-facing*
  - In each row, the animation proceeds frame by frame from left to right until it reaches the rightmost edge of the sprite sheet.
- <u>Frame Dimensions</u>
  - Rest animation frames are 64 pixels wide and 112 pixels high.
  - Walk animation frames are 96 pixels wide and 112 pixels high.
  - Attack animation frames are 128 pixels wide and 112 pixels high.
    - ♦ Walk animation frames are wider to account for the movement involved vis-a-vis

resting; and Attack animation frames are even wider to account for the more dramatic movements involved in a physical attack.

        - NOTE: The game automatically calculates the number of frames in an animation by finding the width of the sprite sheet and dividing by the standard frame width for that animation; the game also grabs each frame of the animation by reference to the standard width, so it's *very important* that your frames be spaced correctly! For the same reasons, it is also very important that every row contain the exact same number of frames.

- Sprite Placement and Dimensions
    - Characters should *never* be designed to be wider than 64 pixels or taller than 96 pixels when at rest.
    - In the rest sprite sheet, make sure that no part of the character protrudes into the top 16 pixels of the frame; if it does, it will get cut off.
    - Standing upright, characters in Telepath Tactics are generally about 56-62 pixels tall from the bottoms of their feet to the tops of their heads.
    - With few exceptions, the character should be centered horizontally in each frame.
- Team Coloration
    - The game interprets certain shades of violet as **primary team colors**. If you use those shades of violet in your character sprites, the game will automatically palette swap those colors to match the colors of whatever army the character belongs to.
        - Colors with the following hex values will be auto-palette swapped to primary team colors. From lightest to darkest:
            1. 8760D2
            2. 713FCF
            3. 6D00FF
            4. 5226BA
            5. 2E1A6B
            6. 1E183A
        - Certain shades of lavender will also be palette-swapped to primary team color variants. From lightest to darkest:
            1. 9FA1D0
            2. 9A85FF
            3. 9183AF
            4. 6C5A81
    - The game interprets certain shades of red as **secondary team colors**.
        - Colors with the following hex values will be auto-palette swapped to secondary team colors:
            1. C84C4C
            2. B93C3C
            3. 8A2E2E
            4. 5E2121
            5. 401717



- Number of Frames
  - A Rest animation must be *only* 1 frame.
  - A Walk animation should be *exactly* 8 frames and loop seamlessly.
  - A Hurt animation should be roughly 5 frames.
  - Attack animations should be *at least* 8 frames long.
    - If the attack is a Move skill with *moveType* Teleport, it needs to be *at least* 12 frames long. No matter how long the animation is, the final 8 frames will always be treated as the "reappearing" portion of the animation.
- Naming
  - Each sprite sheet must obey the following naming convention: *SpriteType_AnimationType.png*
    - The "SpriteType" is a name that refers to a complete set of animations for a given character. (This corresponds to the **spriteType** property used in character classes; *see **Section B2*** above.)
    - For a non-attack animation, use one of the following for AnimationType:
      1. *Rest*
      2. *Walk*
      3. *Hurt*
    - For an attack animation, AnimationType should *exactly* match the name of the attack--otherwise, it will have to be something like *Cast*, to be used with a character's **defaultAtkAnim** property (*see **Section B12*** above).
      - Usage example: *BronzeGolem_Walk.png* tells the game that this sprite sheet contains the movement animation for the BronzeGolem sprite type.
      - Usage example: *Crossbowman_Crossbow.png* tells the game that this sprite sheet contains the Crossbow attack animation for the Crossbowman sprite type.
- Directory
  - The game looks for animations in specific subdirectories. If your animation is in the wrong folder, the game won't be able to find it!
  - Non-attack animations reside in their own folders within *Data* > *Characters*.
  - Attack animations reside in their own folders within *Data* > *Characters* > *Attacks*.
  - Each folder name must *exactly* match the name used for the AnimationType.
    - So, for example: let's say that you created a custom animation for a Swordsman attack named BladeStorm. You'd need to save the sprite sheet as *Swordsman_BladeStorm.png*, and you'd need to save it in *Data* > *Characters* > *Attacks* > *BladeStorm.* If you decide to make this attack available to another character that doesn't use the Swordsman sprite type, you'd change the filename to reflect the new sprite type, but you would still save the new animation in *Data* > *Characters* > *Attacks* > *BladeStorm*. Basically: it's the animation type that

determines the folder it goes in, not the sprite type.

M. <u>Visual Effect Sprites</u>

- Visual effects use sprite sheets to animate, much like characters do.
- <u>File type</u>
  - As with destructible objects and characters, you must use *.png* files with transparent backgrounds.
- <u>Orientation</u>
  - Every visual effect animation is organized into four rows, each representing a different direction the effect might be facing. From top to bottom, these are:
    1. *Down-facing*
    2. *Left-facing*
    3. *Up-facing*
    4. *Right-facing*
  - In each row, the animation proceeds frame by frame from left to right until it reaches the rightmost edge of the sprite sheet.
- <u>Frame Dimensions</u>
  - Visual effect animation frames can be of any size, but they must always be perfectly square (i.e. the height and width of each frame must always be equal). So you can have a visual effect animation with 128 x 128 sized frames, or 64 x 64, or 32 x 32, or even 8 x 8—but never, say, 128 x 96.
    - ♦ NOTE: The game automatically calculates the number of frames in an animation by finding the width of the sprite sheet and dividing by one fourth of the height of the sprite sheet; the game grabs each frame of the animation by reference to the that calculated frame width, so it's *very important* that your frames be spaced correctly! For the same reasons, it is also very important that every row contain the exact same number of frames.
- <u>Sprite Placement and Dimensions</u>
  - With few exceptions, the visual effect animation should be centered both horizontally and vertically in each frame, and should remain in that same spot in every frame.
- <u>Number of Frames</u>
  - A visual effect animation may be any number of frames in length.
  - Bear in mind that if an animation is intended for use as a projectile, the length of the animation will affect how long the projectile takes to travel to its target. A single animation frame translates to 3 frames of travel time in the game's engine; it is recommended that very fast-moving projectiles like arrows and crossbow bolts be limited to a single frame.
- <u>Directory</u>
  - The game looks for visual effects animations in a specific subdirectory: *Data > Characters > Attacks > _VFX.*

N. <u>Procedural Level Generator Room Sets</u>

- When you use GENERATE_RANDOM_LEVEL, you aren't just limited to the types of rooms that come with the game; you can make your own!

- To create a new type of procedurally generated level, create a new subfolder within *Maps* > *Generator Chunks*. The name of this folder is what you'll use in the first parameter of GENERATE_RANDOM_LEVEL if you want to create a level using this folder's chunks.
- Begin creating 7 x 7 square room chunks in the map editor, then save them in your new subfolder.
  1. You *must* have a chunk called **Entrance.xml**. This is the only room chunk that will *always* be used when the game generates a level. Because it's the only one that's guaranteed to be used, it should have spawn locations for the player's characters (i.e. *FromPlayerRoster*, team 0). For the same reason, if there are any scripts or dialog that you want to run every time a level of this sort is generated, you should stick them in *Entrance.xml*.
  2. In **all other rooms**, scatter *FromPlayerRoster* spawn locations for team 1 at every spot where an enemy can potentially spawn; the game will select which of these to actually spawn enemies in at random, based on how many enemies it is told to spawn in its GENERATE_RANDOM_LEVEL instructions. The types of enemies it spawns, likewise, will be determined based on GENERATE_RANDOM_LEVEL instructions.
     - If there are room chunks that you want to have special scripted events or dialog, you can stick the relevant Script and Dialog code within that room chunk. Be aware, however, that the game will combine these with any scripts or dialog present in other rooms it uses as well, so try to avoid duplicate dialog triggers!
- Finally, copy-paste the *_Settings.xml* file from one of the other Generator Chunks subfolders into the new subfolder. Open the freshly copied file in an XML editor; you should see the following six XML tags:
  1. **RoomsDensity** – a number that determines how many rooms spawn relative to the level's size. With a room density of 1, the game will spawn one non-entrance room for every 7 squares of the level's dimensions; with a room density of 1.5, it will spawn 1.5 non-entrance rooms for every 7 squares of the level's dimensions (rounded to the nearest whole number); with a room density of 2, it will spawn two non-entrance rooms for every 7 squares of the level's dimensions; and so on.
     - Usage example: **<RoomsDensity>1</RoomsDensity>** tells the game that in a level with a *level size* of 21, there should be three rooms (21 divided by 7 = 3) in addition to the entrance; that with *level size* 28, there should be four rooms (28 divided by 7 = 4) in addition to the entrance; and so on.
  2. **CorridorWidth** – tells the game how many tiles wide the corridors connecting the rooms should be.
  3. **FloorTile** – tells the game what type of tile to use when generating the floors of connecting corridors. (My advice would be to simply copy-paste this from a room chunk xml file generated by the map editor.)
  4. **FloorTileset –** tells the game which Tiles subfolder the first number in the FloorTile references.
     - Usage example: **<FloorTileset>03,app:/Tiles/Dungeon</FloorTileset>** tells the game that every tile from set "03" resides in the Dungeon tiles subfolder. It is strongly recommended that you simply copy this from the <Tileset> tag in your rooms so that your tileset references remain consistent across the generated level.

5. **WallTile** – tells the game what type of tile to use when generating the downward-facing walls of connecting corridors in an indoor level. Here, too, I'd advise copy-pasting this from a room chunk xml file generated by the map editor.
6. **WallTileset** – exactly like FloorTileset, but for the WallTile.

O. <u>Misc.</u>