Themes of *Caedes Ludi*: Brutal and Scary

Game Flow

Beginning the game from the main menu, players click the start game button and begin playing. A level or series of levels will explain how to play to play through simple obstacles and easy enemy placement. Then, the real levels will begin and the player will have to beat them to progress. Finally, when the player beats all the levels, they get to the end level, where they cause an explosion of some kind to disrupt the ritual as the werewizards are preparing the final rites and escape.

Game States

* Main Menu
* Normal Gameplay
  + Playing
  + Paused
  + Dead
* Final “Cutscene”/Credits

World

Levels based on the outskirts/abandoned parts of Chicago, probably in generic and spooky looking abandoned warehouses/basements in warehouses.

Player

Starts the game as a human, but can transform into a werewolf at the press of a button. Changing from human to werewolf or vice versa uses up a transformation charge, which the player a max of (and starts with) three charges. To regain their charges, the player must either go to the next level, restart the current one, or pick up Potions of Restoration that are scattered around the level.

While in human form, the player can interact with crates/barrels to pick them up and move them around. The player can jump on top of these objects to reach higher areas, but while carrying them the player cannot transform. Player can jump while holding them. Human form can also jump an adequate straight vertical distance upwards and climb ladders. Human form cannot defend itself with any way to attack.

While in wolf form, the player cannot pick up crates/barrels but can still jump on top of them. When the jump key is pressed, the player will leap forward a large horizontal distance forward. If the player collides with an enemy while leaping, they will begin mauling them most of the time. The player cannot stop mauling the enemy until they are dead, which takes a few seconds of button mashing to achieve.

Enemy

There are three proposed types of enemies so far: a human form guard with a gun who patrols across their platform (most basic, highest priority to develop), a human form guard equipped with a hunting rifle who moves through the whole level while tracking the player down (much more complicated, second highest prio), and a wolf form guard who patrols across their platform (somewhat complicated, lowest prio).

All the enemies will have the same two kinds of player detection: cone of vision projected from their front and a radius of hearing around them. The exact dimensions and length of the vision cone may vary on the type of enemy, as could the radius of hearing. Upon spotting the player, the two enemies with guns would pause briefly while playing a “taking aim” animation or something, then fire their guns with a raycast. If the player is hit with the raycast, they die instantly. For the wolf type enemy, when it sees the player it will run to within leaping radius and then leap at them. If the enemy wolf hits the player, they die. The enemy wolf’s leap has higher priority than the player’s leap, meaning that if they both leapt at each other at the same time, the enemy wolf would kill the player.

If any enemy sees a player and then their vision of them is broken, they will run along their platform towards the direction of the last place they saw the player. If they reach the edge of their platform and have not found the player, they will stand still for a few seconds while playing a “confused” animation, then resume patrolling or hunting. The hunting enemy may behave differently when losing sight of the player.

Interactable Objects

* Crate/Barrel: Can be picked up by the player while in human form and placed elsewhere. Can jumped on top of. Provides cover for player from enemy vision/shots. Stops enemies short during their patrol if they collide with it (maybe).
* Ladder: Can be climbed up or down by player while in human form.
* Door(?): Interacting with this could move the player to a different door in the same level, through an implied hallway/stairway.
* Potion of Restoration: When player interacts with it, restores all their transform charges.

Level

Levels are split by triggers at the edges of the level. Camera is shifted to new view immediately on level transition, otherwise follows player smoothly. On entering new level, player’s transform charges are refreshed. Any enemies aggroed on player will despawn/deactivate. Checkpoint system maybe?

The Furnace

To prevent extra assets / scripts creating confusion in the active folders, I’ve created THE FURNACE. It’s essentially The Recycling Bin. If you replace a temp asset, or refactor a script to the point its basically a new file, place the old copy in the furnace. This way we don’t have to worry about removing something someone might actually need in the future, but is not implemented at the moment.

Character Animations

There are a number of elements required for the state based animations of the in game characters. First off, it requires the Spine - Unity Package [available here](http://esotericsoftware.com/spine-unity-download) to import and manage the animated assets created in Spine.

**Preferences**

Under File>Preferences>Spine there are a number of options related to the import plugin. Most notable is the Default SkeletonData Scale which controls how the import is resized from the native Spine file. Resizing this asset in engine can create physics / animation difficulties.

**Additional Folders**

Spline added its dependencies to two folders within the Assets folder. They are Editor and Spine.

**Assets**

As there are a number of assets associated with each character in the game, I created a folder called Animations. Within this folder will be subfolders corresponding to each player character. In this particular case the players human and wolf forms will be in separate folders.

name\_SkeletonData - This is the main Unity file that was generated by the plugin. It stitches together all of the data stored in the imported JSON and all of the assets. This asset contains a button which generates the state controller containing each animation.

name\_Controller - This is the animation state controller containing all of the animations associated with the character. See breakdown of state controller below.

Name.json - This is the file exported by Spine software that stitches together the image and the atlas. This file when imported into Unity becomes the \_SkeletonData.

name.atlas.txt - The atlas files contain position information for each of the image segments that are spliced together as well as each of the bones for the skeleton.

name\_Material - This is an image file with all of the sub images of the character laid out similar to a sprite sheet.

**State Controller**

The connections between each animation are created with a logical conditional and can be called via script. Most of them are managed via a boolean value. NOTE: It is of importance if you want to call an animation from script you should call the *transition* rather than the animation itself. Otherwise the controller will no longer have control over that instance of animation. This also allows for the blending between the animations to be applied to your call.

To call from script you will need a reference to the <Animator> component

Most notable methods for me so far are:

.SetBool(“NameOfTransition”, boolValue);

.SetTrigger(“NameOfTransition”);

Rapid Prototyping

To help speed up the process of iterating on level design I created a simple tile map with colliders so that you can draw out black tile primitives in engine. These will ultimately serve as the actually games colliders and will be stored on their own Sorting Layer. We can then cover these with another layer of finalized assets to create a stylized environment.