Design Document

Mild Goose Chase

(Video Game)

By: Kyle Kaminski, Franklin Adair, Matt Keville, Victor Caceres, Chris Falotico, Timothy Gitt, John Miller

Project Leader / Scrum Master: Kyle Kaminski

Professor: Dr. Baliga

Github: GitHub

Slack: Senior Project

Google Drive: Google Drive

Table of Contents

Description

This application is a fully-functioning computer video game for the user to play. It is built off of one of the latest builds of the Unity game development engine (as of Spring 2020). The game will function as a procedurally generated "dungeon-crawler" roguelike game where the player will progress through randomly-generated dungeons, fighting different types of enemies along the way. Their goal is to kill a specified number of enemies and find the exit somewhere in the level. The user will take control of the player character and navigate the dungeons to progress as far as possible. Should the player perish, the game will end and will need to be restarted from the beginning. Should the player find the exit, the game will advance to the next level.

The user will control the player with a mouse and keyboard. The player will be able to move and attack enemies using melee (very close-combat with a sword) attacks. When an enemy is attacked, it will take a certain amount of damage based on the player's and enemy's attributes. When an enemy's health reaches zero, it will die; the same will go for the player as well if the player's health reaches zero. If the player's health reaches zero, the game will end.

There are many features present in this game to make it unique in the industry. It will be free to play for anyone to enjoy. Movement and attack controls, melee combat, ranged combat, AI-driven enemies, procedural level generation, and a goal to progress as far as possible will allow the user to play and enjoy the game. Other features include a graphical user interface (GUI), pause menu, impassable terrain, proceeding to the next level, and other player attributes. The details on most of these features will be outlined in great detail throughout this document.

Screen

The screen is the visible aspect of the game that the player will view for the duration of the game. The screen will show menus, the HUD, player and enemy attributes, and the game itself, which includes the player, enemies, and world. The screen will be zoomed out to the point of not being too close to obstruct the vision of the player, and not zoomed in to the point of not being too far where the player can see most of the level, giving them too much of an advantage (ex. Seeing an item in

the next room, so the player skips ahead to that room instead of trying to clear the enemies out of the current room).

The screen movement will be fixed, instead of locked. A fixed screen means that the scrolling of the screen will be locked in a way that the player will always be in the center (screen moves 1:1 with the player). A locked screen means that the current room the player is in fills up the entire screen, and does not scroll with the player (screen is more zoomed out and player is not locked to being the center of the screen at all times; think like a dungeon in the original The Legend of Zelda on the NES). A fixed screen will ensure that the player is not overwhelmed with too many things in the room by seeing too many things at once, and will only see enemies and items that will be close enough to interact with at that point in time.

HUD

The HUD (Heads Up Display) of the game is the collection of useful player meters that are on the screen at all times. Some of these meters are health and stamina for the player, and health for the enemies. Having the HUD not only on the screen at all times, but showing useful information at all times, will let the player know how they should play the game in certain situations. For example, if the player has a high amount of health and they see an item and an enemy at the same time, they may attack the enemy first. However, given the same exact situation but with a low amount of health points, the player may play more defensively, as the enemy poses a bigger risk to the player. The same applies to stamina: the player may choose to attack enemies when they have higher stamina, but may avoid them until their stamina recovers. The player can also press the “K” key to pull up a menu of all their attributes, so they can see their current standing in the game. The HUD provides this valuable information to the player in these scenarios, letting the player decide how they want to approach the game in their own way at certain times.

Map

The game will feature a fully fleshed out procedurally generated map that will be unique for every level and playthrough. The map will contain rooms that are placed together using a procedural generation algorithm so they are in a different overall layout for every playthrough. When the level is completed and a new level is constructed, a new level will be constructed again using the same algorithm.

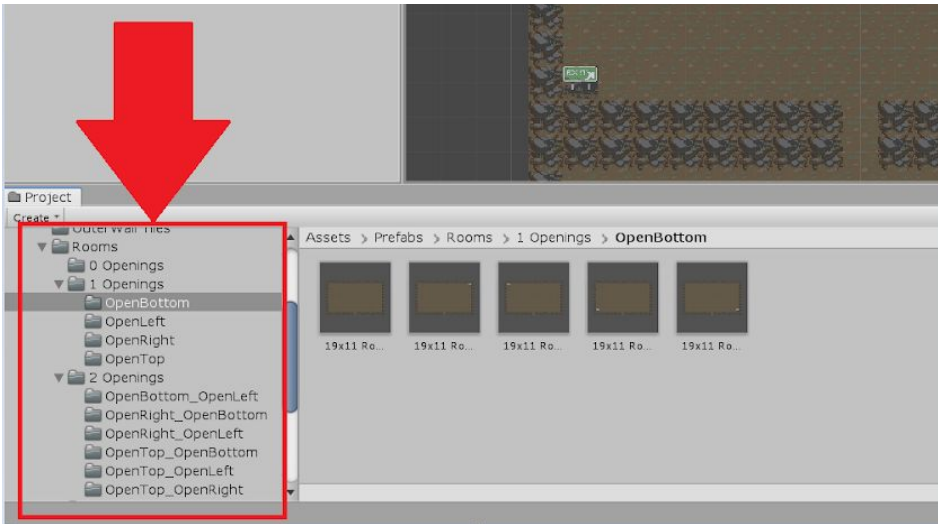
Theoretically, this will be unique every time.

The game will have this feature to differentiate it from other games and to give the player near infinite replayability. When using predetermined levels with the same enemy spawns, item spawns, level design, etc., the player knows what to do and where to go every time. Using procedural generation in our game allows the player a familiar yet unique experience every time.

Prototype Image Below (What an average room may look like after being constructed, subject to change):

The game will have predetermined sprites, tiles, enemies, enemy pathfinding and behaviors, etc. These rooms will be constructed into a unique layout upon each game/level creation. The procedural room generation will ensure that although there might be similar rooms, they may not have the desired level exit or enemies in it that will deem it useful to the player.

Prototype Image Below (Room prefabs)



Player

The player is the game object that the user has control of at all times. This game object has specific attributes that may change over time as the game progresses. Two major attributes of the player are its hitbox and its hurtbox.

The player hitbox is the area around the player where damage to the enemy can be given. For example, the player's hitbox in our game will be its sword swipe. The player hurtbox is the area around the player where damage from the enemy can be received. For example, this is the player's sprite.

The player interacts with other game objects in the game with the help of the Unity Engine and backend C# scripts. Some examples of the types of interactions that may happen can be (but not limited to): hitbox vs hitbox, hitbox vs hurtbox, and hurtbox vs hurtbox. These interactions are all handled by scripts and are core mechanics in the player/battle system of the game.

Controls

The user can control a character with the controls on their keyboard. The user can make the character move and attack. There will be keys that will guide the player in the four cardinal directions that the keys indicate. For example, the up arrow key will move the player upwards, down arrow key will move them downwards, and so

on. The user can also combine nearby cardinal keys to go in the four ordinal directions as well. For example, clicking the up arrow key and the left arrow key will let the player go northwest. If the player has a gamepad controller that can bind keys to its inputs (ex. Valve's Steam Controller), then the user can play the game with a controller, in theory. This will allow for greater ease of use and enjoyment for the user.

The user can attack and move using the controls provided to them. There are many different attacks in the game that the user will experience using. By using the designated [Attack] button, the user can have the player perform an attack. This is also planned to be mapped to a gamepad such as a Dualshock 4 or a Pro controller. The player also gets a dash attack by using the "Shift" key, where the tradeoff is some stamina points.

Enemies

There will be a multitude of enemies that the user will fight in order to progress through the game. These enemies will try to kill the player using similar mechanics that the user has access to. Like what was mentioned in the Player section, enemies will be the main point of interaction of the hitbox and hurtbox mechanics.

The types of enemies in the game are: the Slime (and mini Slime), the Butcher, Sparkacus, and the Archer Nemesis. Each enemy serves a unique purpose to diversify the gameplay. The slime is a standard enemy with a small attack range and slow movement, making an easy target for the player to kill (think Goombas in 3D Super Mario games). The Butcher is another melee-based enemy with an even slower movement speed, but has much more attack range and attack damage than the slime. Sparkacus is a range based enemy that does not move much, but shoots off constant bursts of very strong projectiles; this means the player needs to time their movement correctly in order to avoid the attacks and strike the enemy when they are not attacking. Last is the Archer Nemesis, who is similar to Sparkacus, but they have a higher chance to move away from the player, and their attacks are not as strong.

All enemies also have what we define as "desperate" attacks (similar to enemies in the "Dragon Quest" series). When an enemy is being attacked, they have a chance to perform a sudden desperate attack, which is another greater threat the player must now worry about. The Slime's desperate attack is to split into multiple

mini Slimes with much faster movement speed. The Slime can split into any random amount of mini Slimes. The Butcher's desperate attack is to be more aggressive towards the player. Sparkacus has a higher rate of fire as a desperate attack. Finally, the Archer Nemesis' desperate attack is to flee to a corner of a room, and/or increase their rate of fire, making them much harder to kill.

For player hitbox vs enemy hitbox (attacks clashing), the player may take damage from the enemy. For player hitbox vs enemy hurtbox (player attacks enemy), the enemy will take damage from the player. For player hurtbox vs enemy hitbox (enemy attacks player), the player will take damage from the enemy. For player hurtbox vs enemy hurtbox (player and enemy objects clashing), the player will take damage from the enemy.

We must also consider the enemy behavior when interacting with other enemies, which is also a core mechanic for enemy interaction. For enemy A hitbox vs enemy B hitbox (attacks clashing), the enemies will take damage from each other (although they may not target each other, they can damage each other through stray attacks). For enemy A hitbox vs enemy B hurtbox (enemy A attacks enemy B), enemy B will take damage from enemy A. For enemy A hurtbox vs enemy B hitbox (enemy B attacks enemy A), enemy A will take damage from enemy B. For enemy A hurtbox vs enemy B hurtbox (enemy A and enemy B objects clashing), enemy A will take damage from enemy B.

Another feature of the enemy object in the game is that the enemies will be generated through an evolutionary process that makes them more challenging for the player as they play the game. This process will evolve enemies by having the enemies evolve with respect to the player (ex. When the player moves to the next level).

The enemies will also consist of a finite state machine to control how they will behave in game, and how they interact with the entities and game objects in the scene. Some examples of enemy behavior in the finite state machine are (but not limited to): enemy vs player interaction, enemy vs enemy interaction, and enemy evolution.

Enemy AI: The Finite State Machine

Enemy entities consist of multiple states, which is used to determine what functions should be used without having them operate when it is unwanted. For example, we would not want the enemy to be chasing a target if it is knocked back, since the chasing movement will cancel the knockback movement, and make it seem like it was never knocked back.

States included are as follows:

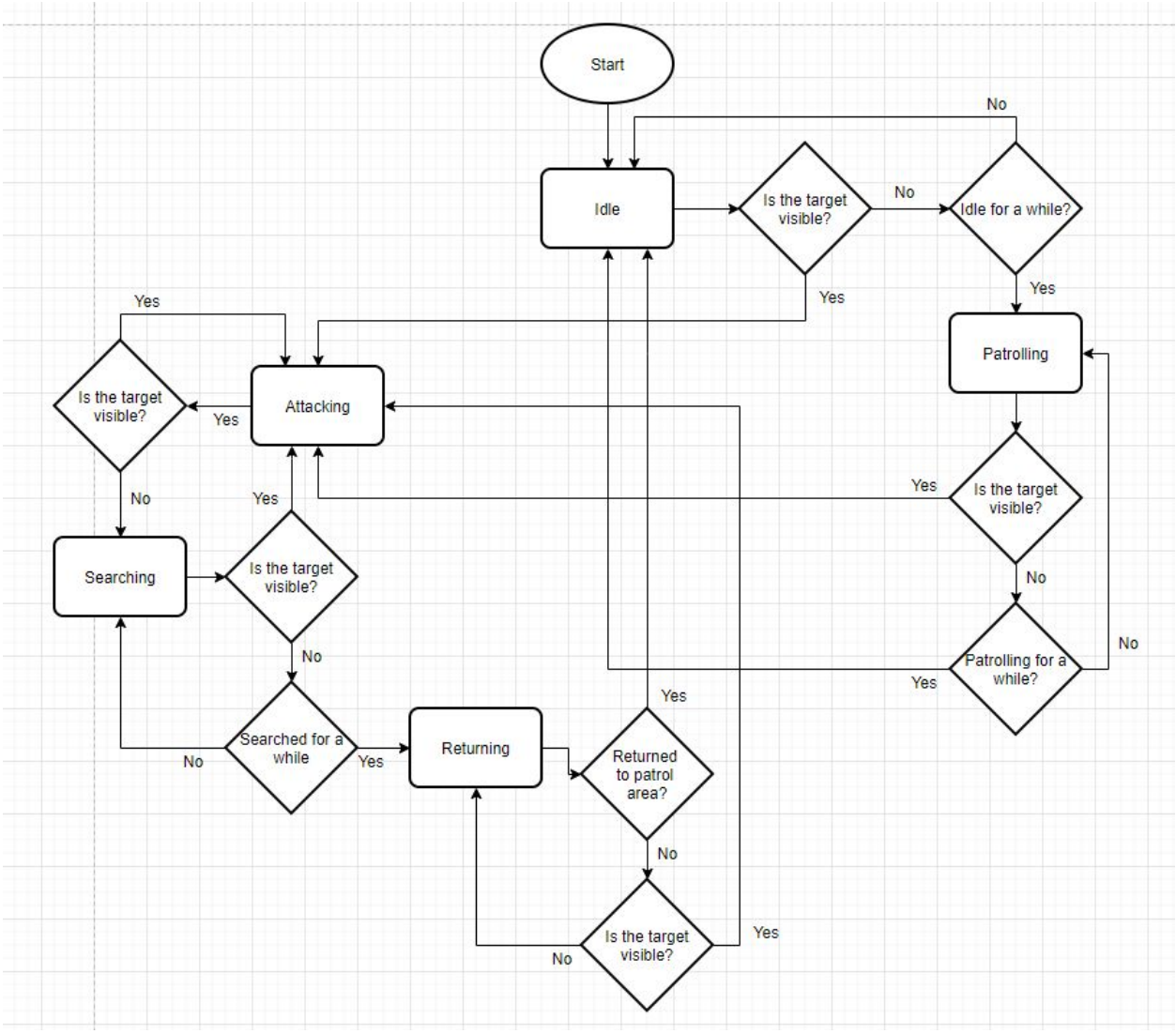
- Idle
- Patrolling
- Attacking
- Searching
- Returning

The Decision Tree:

A decision tree is created in order to facilitate the understanding of what enemies in the game should do under certain conditions. When an enemy is in a certain state, it will run functions associated with that state and then check conditions to determine what state it should be in after an update.

Decision trees are not the same for all enemy entities. Therefore, different types of enemies will behave differently and have a different decision tree, but will share from a shared pool of states.

Below is an example of a decision tree for a simple enemy (Slime), that has a basic melee attack and will chase the target if it gets close. This decision tree may not fully represent the final build of the game.



Melee

Melee refers to a short/ medium ranged attack that both the player, and certain enemies have. This attack will deal damage to whatever entity (either player or enemy) it collides with and knocks that entity back. After an entity is struck by a melee attack they will be put into a state in which they cannot take damage for a

short amount of time. The direction of melee attacks will depend on which way the entity is facing; for instance, if an entity is facing left and they trigger a melee attack, then the animation will be instantiated towards the left. The available directions are left, right, up and down.

Prototype Image Below (Visualization of what an AI may see when a player is in view):

Ranged

A range attack is an attack that consists of a projectile entity that can collide with other entities and deal damage and/or knockback. This attack will be available to both the player and specific enemies and bosses. This projectile is instantiated from the entity that uses the attack. The projectile is a prefab that consists of a damage variable and a hitbox. Projectile type will vary depending on the entity that instantiates them (e.g. a wizard will launch a magic missile or an archer will shoot an arrow). The direction of ranged attacks will depend on which way the entity is facing; for instance, if an entity is facing left and they trigger a ranged attack, then a projectile will be instantiated moving left. The available directions are left, right ,up and down.