Brandon Maxwell

ComS 437

Overall Game Architecture

My overall design uses normal Object Oriented Design principles. I would much rather use either the Differentiated Components method or the Undifferentiated Components method, however, I have no idea how to go about designing a game using these systems. The reasons why we would want to use either of those ideas is understandable, but I have no idea how to apply the approaches to my own game, so I decided to design the game using what I already know.

Character Hierarchy

ICharacter Interface

- An Interface that defines some common behavior that all Characters need. In my game all characters can be updated and drawn. It is up to implementing class to define how this behavior works.

Character Class

- The Character class will define a good chunk of the behavior that all subclasses use. Characters will be drawn using the model that they hold as a field, using their current world matrix. All characters also have two different forms of attack: range and melee. As such all characters will hold both a range weapon and a melee weapon as an instance field. Character also provides a protected move method that subclasses can implement. This allows the move method to be different depending on whether the character is a user controlled object or an AI controlled object. Characters also have a certain amount of health, and when that value reaches zero, the character will be removed from the game. By implementing the

Collidable interface, characters have the ability to check if they are colliding with another object.

PlayerCharacter

- The PlayerCharacter class represents a Character whose move method behavior depends on the input from the user. Since the Character class defines behavior for drawing and attacking, the PlayerCharacter only needs to define the move method.

AICharacter

- The AICharacter class is the base class for all AI characters. It provides a method makeDecision to stub out the AI decision process. Since minion characters and boss characters need to make decisions differently, this method is marked as virtual.

Minion

- The Minion class represents a generic low level enemy in the game. Minions will override the makeDecision method from their super type.

Boss

- The Boss class is a form of AI that has more functionality than the Minion type. Bosses will have some form of special attack that they will need to decide to use in certain situations. This decision will be made in the makeDecision method.

Weapon Hierarchy

Weapon

- Weapon is the base class of all weapon items. A weapon has a specific state, depending on whether the weapon is in an idle state, an attacking state, or a blocking state. Weapons are Collidable, but the implementation of the collidesWith method is left to subtypes. The implementation of the update method is also left to subtypes

MeleeWeapon

- MeleeWeapon is a subtype of Weapon. This class will define the behavior of the collidesWith method. Melee weapons differ from other types of weapons in that they will be a part of the model for characters.

Projectile

- Projectile extends from the Weapon class. Since range weapons are not a part of the model for characters, they need to have their own model and their own draw method. Projectiles also need to have Matrix for their world, which will hold the weapon's current position

MagicSpell

- A MagicSpell is a type of projectile. This type of projectile is not affected by gravity so its update method is implemented to reflect that. Other basic functionality is implemented in super types.

Arrow

- Arrows are a type of projectile that are affected by gravity. Their update method with also need to be implemented to gravitate toward the ground. Their other behaviors are implemented in super types.

Game Class and Components

Game

- The Game class is the default auto-generated Game class from XNA. It will hold a camera, to use for viewing the scene, and two Managers. The first manager is the CharacterManager, which handles all the information and processing involving characters. The second Manager is the LevelManager, which handles identifying the level the player is currently in.

CharacterManager

- The CharacterManager is a class that will handle all interactions between the characters in the game. It has two fields for the user controlled characters: player1 and player2. The game will typically need to treat these specially since users can drop in and out of the game arbitrarily. Also as a field in the CharacterManager is a list of Characters representing the enemies found in the level. The CharacterManager is responsible for updating, drawing, and resolving collisions found between the characters.

LevelManager

- A LevelManager manages the current level the player is on. To be honest I'm not entirely sure what this entails. This class holds a Level object as a field, which

represents the current level the user is playing.

Level

- The Level object is a representation of the current level the user is playing through. Since the game is a platformer style game, where the user only has two degrees of movement, the level is held as a 2D array of Tiles put on top of a background image. Tiles can be either platforms, which are solid, or open space.

Tile

- A Tile is an object that boxes up an Image and a location. The image is what will be drawn to the screen, with different images for platforms and open spaces. The location field of the tile represents its column and row number in the Level's 2D array.