Entity Classes

`Interface WorldEntity`

Responsibilities

- Position: Attribute, double[x,y], continuous coordinates for where the entity is located
- Shape: Attribute, the outline of the entity

Collaborators

Player, Enemy, Defender

`abstract class GameCharacter`

Responsibilities

- Health: Attribute, int, denotes the total amount of health the individual has
- Position: Attribute, array with x, y, where on the map the IC is currently located
- Inventory: Attribute, ArrayList, items currently being held
- Level(?): Attribute, the level of the enemy
- Range(?): Attribute, how far the character can attack
- Hitbox(?): Attribute, from how far the character can be hit from

Collaborators

Player, Enemy, Defender

`class Player extends GameCharacter implements DamageableCollidable`

Responsibilities

- represents the `GameCharacter` controlled by the player

Collaborators

Player, Enemy, Defender, Collidable, GameCharacter

`class Enemy extends GameCharacter implements DamageableCollidable`

Responsibilities

- represents enemy GameCharacters which attack the Player's castle and defenders

Collaborators

GameCharacter, Player, Defender, Map, DamageableCollidable

`class Defender extends GameCharacter`

Responsibilities

- represents friendly GameCharacters which attack enemies

Collaborators

GameCharacter, Player, Enemy, Map, DamageableCollidable

`class Tile`

Responsibilities

- describes how a tile will appear and behave on the map
  - texture: what the tile looks like (i.e. grass, road, water, etc.) – could be path to image file
  - isCollidable: boolean, can other entities collide with this tile
- has dict for metadata specific to certain tiles
  - spawn_point: boolean, is this a spawn point for the enemies

Collaborators

World, Map

`class TileEntity implements WorldEntity`

Responsibilities

- WorldEntity representing map Tiles in-game.

Collaborators

World

`class Map`

Responsibilities

- `tiles`: `Tile` objects used in this `Map`
- `layout`: Attribute, 2d array of tiles

Collaborators

Enemy, World, Tiles

`class World`

Responsibilities

- entities: holds all WorldEntities

Collaborators

Map, WorldEntity

`interface Collidable`

Responsibilities

- Ensures that no two objects implementing Collidable are at the same place at the same time.
- Collidable objects have an onCollision method which defines their behaviour upon collision.
- Collidable objects have a hitbox, their shape in the world of Collidables.
- (If necessary) label system to ignore collisions with certain objects

Collaborators

Player, Enemy, Defender

`interface DamagingCollidable extends Collidable`

Responsibilities

- Has information about how much damage to inflict when it hits something, who caused the damage

Collaborators

Player, Enemy, Defender

`interface DamageableCollidable extends Collidable`

Responsibilities

- When a DamageableCollidable collides with a DamagingCollidable, the DamageableCollidable (or manager's) takeDamage method is called.

Collaborators

Player, Enemy, Defender

`abstract class Weapon implements Item`

Responsibilities

- behaviour determined by `WeaponUsageDelegate`
- level: item metadata, current upgrade level of weapon
- static dict with attributes for each level
  - damage
  - range
  - attack speed (?)

Collaborators

Player, Enemy, Defender, DamagingCollidable

`class ControlsState`

Responsibilities

- is the common language representing states of a GameCharacter's input device (keyboard, controller, etc.)
- uses `double` attributes for Up/Down, Left/Right, and other inputs

Collaborators

KeyboardInputHandler, GameCharacter

`abstract class LevelState`

Responsibilities

- represents the state of a level
- Stores static information about the level
  - Map
  - Duration
  - Which/When/How many enemies spawn
- Can also store dynamic information about a level being played
  - World
  - Time passed

Collaborators

World, Map, GameCharacter

`abstract interface Item`

Responsibilities

- store information about an Item
  - texture
  - name
  - metadata: misc. information not common to all `Item`s
- has an `ItemUsageDelegate` which determines the behaviour of this item when used

Collaborators

ItemUsageDelegate

Use Case Classes

`class ItemUsageDelegate`

Responsibilities

- `ItemUsageDelegate`s must implement a `use` method which takes an `Item` and the `GameCharacter` that used it.
- `use` does nothing by default

Collaborators

Item, GameCharacter

`class WeaponUsageDelegate implements ItemUsageDelegate`

Responsibilities

- spawns in a DamagingCollidable which actually inflicts the damage for the weapon used

Collaborators

Weapon

`class CharacterManager`

Responsibilities

- Animate the character based on inputs
  - Inputs from some `InputManager` stored as a `ControlsState`
- Responsible for what happens upon certain inputs
  - moveCharacter: update character's position
  - useItem: method, activates an inventory item
  - addInventory(item): add item to inventory
  - removeInventory(item): remove item from inventory
  - openInventory: method, returns inventory contents (use presenter to display)

Collaborators

Player, Enemy, Defender, Collidable

`class GameManager`

Responsibilities

- beginGame: trigger process to begin a game
- deleteCharacter: removes a character from the map when health = 0

Collaborators

Player, Enemy, Defender, Map

Controller Classes

`class LevelManager`

Responsibilities

- Initialize level's `World`
  - convert `Map` into `TileEntity` objects to add to the `World`
  - invoke `SpawnController` for the `Player` and other `GameCharacters`
- Query level state
- Reset level to certain `LevelState`
- pause/play/Progress the level

Collaborators

GameManager, LevelState, World, WorldEntity, Map

Controller Classes

`abstract class InputHandler`

Responsibilities

- Get input from any source and return a `ControlsState` representing the input.

Collaborators

ControlsState, GameCharacter

`class KeyboardInputHandler extends inputHandler`

Responsibilities

- Translate keyboard inputs into a `ControlsState`
  - `keyLeft` move player left
  - `keyRight` move player right
  - `keyDown` move player down
  - `keyUp` move player up
  - `keyOpenInventory`, browse inventory
  - `keyChooseInventoryItem`, pick inventory item
  - `keyUseItem` place inventory item
  - `keyLevelUpDefender` level up the defender
  - `keyAttack` attack with weapon

Collaborators

Player, CharacterManager

`class AIInputHandler extends inputHandler`

Responsibilities

- Translates an AI's inputs to a `ControlsState`
  - Since AIs could just generate a `ControlsState`, might just pass it straight through
  - Or, the AI inputs generator could itself implement `InputHandler`

Collaborators

Defender, Enemy, CharacterManager

`class SpawnController`

Responsibilities

- spawnLocation, attribute, (x,y) coordinate, different for player and enemy
- spawn(spawnLocation), method, spawn player/enemy at spawn location

Collaborators

Player, Enemy, Map

Other Classes

`abstract class MenuScreen`

Responsibilities

- handle menus with clickable buttons, text fields, etc.
- position of elements, what happens when clicked, etc.

Collaborators

TBD

`class MainMenu extends MenuScreen`

Responsibilities

- buttons for
  - Start
  - Help
  - Quit

Collaborators

MenuScreen

`class PauseMenu extends MenuScreen`

Responsibilities

- buttons for
  - Resume Game
  - Help
  - Quit

Collaborators

MenuScreen