Entity Classes

`abstract class WorldEntity`

Responsibilities

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

Collaborators

GameCharacter

`class GameCharacter extends WorldEntity`

Responsibilities

- Instantiates character objects such as Player, Defender, Enemy
  - Position and shape inherited from WorldEntity
  - Team: Attribute, determine if Defender, Enemy (who damages who)
  - Health: Attribute, int, denotes the total amount of health the individual has
  - Inventory: Attribute, ArrayList, items currently being held
  - Level: Attribute, the level of the GameCharacter

Collaborators

WorldEntity, World, Map

`abstract 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
  - base: boolean, the base that the player is protecting from the enemies

Collaborators

Map

`class TileEntity extends WorldEntity`

Responsibilities

- WorldEntity representing map Tiles in-game.

Collaborators

World, WorldEntity

`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

`abstract class Weapon implements Item`

Responsibilities

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

Collaborators

GameCharacter, Item

Use Case Classes

`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

CharacterManager, Weapon

`interface DamagingCollidable extends Collidable`

Responsibilities

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

Collaborators

Weapon

`interface DamageableCollidable extends Collidable`

Responsibilities

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

Collaborators

CharacterManager

`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, LevelManager

`class Map`

Responsibilities

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

Collaborators

GameCharacter, World, Tiles

`interface 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, CharacterManager

```
class WeaponUsageDelegate implements
ItemUsageDelegate, DamagingCollidable
```

Responsibilities

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

Collaborators

Weapon, DamagingCollidable, CharacterManager

`class CharacterManager implements DamageableCollidable`

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
  - depleteHealth: decreases characters health when they take damage
  - increaseLevel: increases the level of enemies after completing a wave
  - useItem: method, invokes the use method of ItemUsageDelegate
  - addInventory(item): add item to inventory
  - removeInventory(item): remove item from inventory
  - openInventory: method, returns inventory contents (use presenter to display)
  - OnCollision: method, Collidable interface

Collaborators

`class GameManager`

Responsibilities

- beginGame: trigger process to begin a game
- deleteCharacter: removes a character from the map when health = 0
- world: list of all WorldEntities

Collaborators

GameCharacter, Map, LevelManager, WorldEntity

Controller Classes

`class LevelManager`

Responsibilities

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

Collaborators

GameManager, LevelState, World, SpawnController

`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, CharacterManager

`abstract class InputHandler`

Responsibilities

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

Collaborators

ControlsState

`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

ControlsState

`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

ControlsState

`class SpawnController`

Responsibilities

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

Collaborators

LevelManager

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 menus.PauseMenu extends MenuScreen`

Responsibilities

- buttons for
  - Resume Game
  - Help
  - Quit

Collaborators

MenuScreen