**6.6HD - HD Level Custom Program**

Jayden Kong, 104547242

Program screenshots

A screenshot of a computer

Description automatically generated

The title screen. The player can start a new game here or exit the program

A screenshot of a video game

Description automatically generated

Screenshot of a game, showing all currently implemented entities. In this position, the player can attack the enemies to the top, left and right of them to turn them into coins, or they could move down to collect coins. The red numbers represent health, the blue numbers represent attack, and the yellow numbers represent coins.

A screenshot of a computer

Description automatically generated

The pause menu. The player can choose to resume the current game, start a new game or just quit the game

A screenshot of a computer

Description automatically generated

The death screen. The player’s stats such as coins and kills are shown, along with their final score and high score. From this screen, the player can choose to return to title to start a new game, or quit the game

Program summary

For my custom program, I created an infinitely generating dungeon made up of a grid of cells. The goal of the game is to get as high of a score as possible which can be achieved by collecting coins and killing enemies. Each cell of the dungeon is occupied by an entity such as the player, enemies, coins, or swords.

The player and the dungeon master take turns. During the player’s turn, the player will be able to move their entity to move to one of the four adjacent cells (up, down, left or right) using the directional arrow keys or the WASD keys (if it is a valid move e.g. not outside the 2D array). During the dungeon master’s turn, the dungeon master will be able to move a single enemy into the empty cell left behind by the player. After the dungeon master makes a move, it will fill the empty cells in the dungeon with a new enemy or item, depending on the number of different types of entities that exist in the dungeon.

Depending on the type of entity in the cell the player attempts to move into, the game will react differently. If the player moves to a treasure cell, their coins will increase. If the player moves to a weapon cell, the player’s attack is replaced by the weapon’s attack. If the player attempts to move to a cell occupied by an enemy, the player will not move into the cell, but will instead attack the enemy.

If the player’s attack is higher than the enemy’s health, the player’s attack will be reduced by the enemy’s health, and the enemy cell will be replaced by treasure. If the player’s attack is lower than the enemy’s health, the enemy’s health will decrease according to the player’s attack, and the player’s attack will become zero. If the player’s attack is zero, the player’s health will be used up to deplete the enemy’s health. If the player’s health reaches 0, the game ends.

Class List

* Cell
* DeathScreen
* Dungeon
* DungeonMaster
  + Singleton
* Empty
* EmptyInCell
* Entity
* EntityFactory
  + Singleton
  + Factory
* GameScreen
* HighScoreManager
* IPlayerMovement
  + Interface for strategy pattern
* IScreen
* PauseScreen
* Player
* Potion
* PotionInCell
* ScreenManager
  + Singleton
* TitleScreen
* Treasure
* TreasureInCell
* Weapon
* WeaponInCell

Core classes

**DungeonMaster:** This class manages the game logic. It has methods for moving the player in the dungeon, fetching a player cell and the cell the player needs to move to, calculating player score, executing its own turn, drawing the dungeon game and starting a new game. All interactions with the dungeon need to be done through the dungeon master, since it stores the instance of the dungeon used for the game.

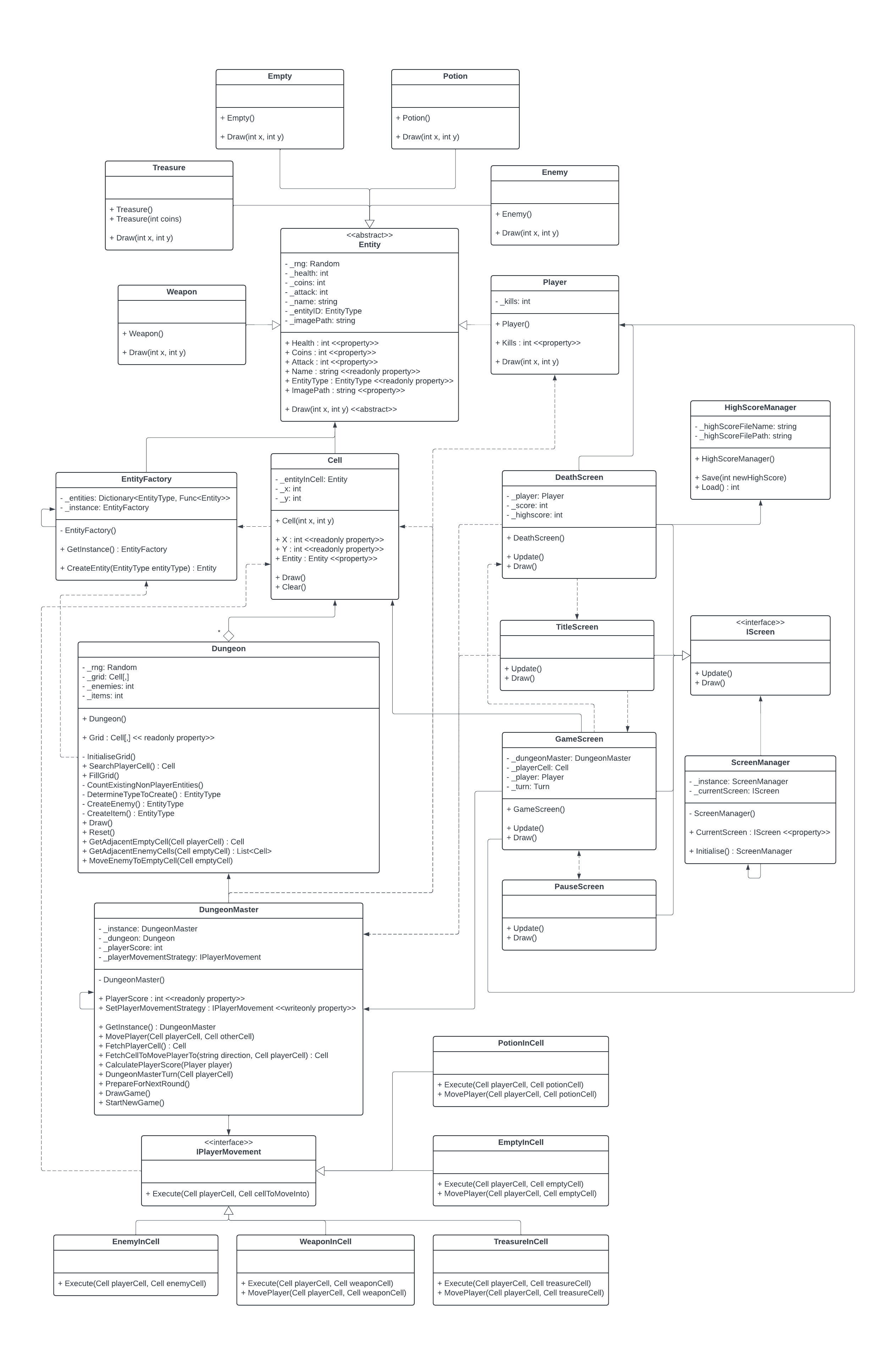
**Dungeon:** This class is made up of a collection of cell objects. The dungeon has methods for fetching specified cells, drawing itself, filling empty cells with new entities and resetting itself to an initial state. The dungeon doesn’t know what entity is inside each of the cells directly and can only be controlled through the dungeon master.

**Cell:** Nine of these cells make up a dungeon. They know their position in the dungeon and contain an entity. They have methods for drawing themselves and for clearing the entity currently contained in them to replace with an empty entity. Cells only know that they contain an entity, but not necessarily what type of entity they contain.

**Entity:** This class is an abstract class which defines the stats that are stored by the entities. Each entity knows its own health, coin value, attack, name, entity ID, and image path. Each class that inherits from Entity sets its own name and entity ID for easy identification (I don’t have to use casting). Each entity has a method to draw its own image as a SplashKit bitmap.

**IPlayerMovement:** This is the interface for the strategy pattern for player movements. The game needs to react differently depending on the entity in the cell the player attempts to move into. To achieve this, a private field for the player movement strategy is stored in the dungeon master and can be changed through a property. Each class that implements IPlayerMovement has an execute method to perform different reactions to the player attempting to move into another cell.

**ScreenManager:** This class stores an instance of an object which has implemented the IScreen interface. The program will tell the stored object in the ScreenManager to update and draw. This implementation allows for a modular screen system, at the expense of increased overhead and an increased number of classes.

**Class Diagram (you may have to zoom in)**

**Sequence Diagram**

This is an example for moving a player into a weapon cell.

A diagram of a cell

Description automatically generated

**Code**

Link to program: <https://liveswinburneeduau-my.sharepoint.com/:f:/g/personal/104547242_student_swin_edu_au/ErU4Y64kH4JHoZLtj1KpAGcBauZ9E1xbm-MP62yfXGPeQA?e=gKNlYv>

Code in its current state will be provided in the following pages.