Design Overview for Dungeon Cells

Name: Jayden Kong
Student ID: 104547242

Summary of Program

Describe what you want the program to do… one or two paragraphs.

For my custom program, I want to create an infinitely generating dungeon game made up of a grid of cells. Each cell will be occupied by an entity such as the player, monsters, treasure, or weapons. The game will be turn based with the player and the dungeon taking turns. The player will be able to move to move to one of the four adjacent cells (up, down, left or right) on their turn. The goal of the game is to get as high of a score as possible.

If the cell the player moves to is occupied by treasure, their coins will increase. If the cell the player moves to is occupied by a weapon, the player's attack will become the same as the weapon's. If a player attempts to move to a cell occupied by a monster, the player will not move into the cell, but will instead attack the monster. If the player's attack is higher than the monster health, the player's attack will be reduced by the monster's health, and the monster cell will be replaced by treasure. If the player's attack is lower than the monster's health, the monster'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 monster's health. If the player's health reaches 0, the game ends. On the dungeon's turn, the dungeon may move the dungeon cells around the player and fill empty dungeon cells.

Improvements for HD:

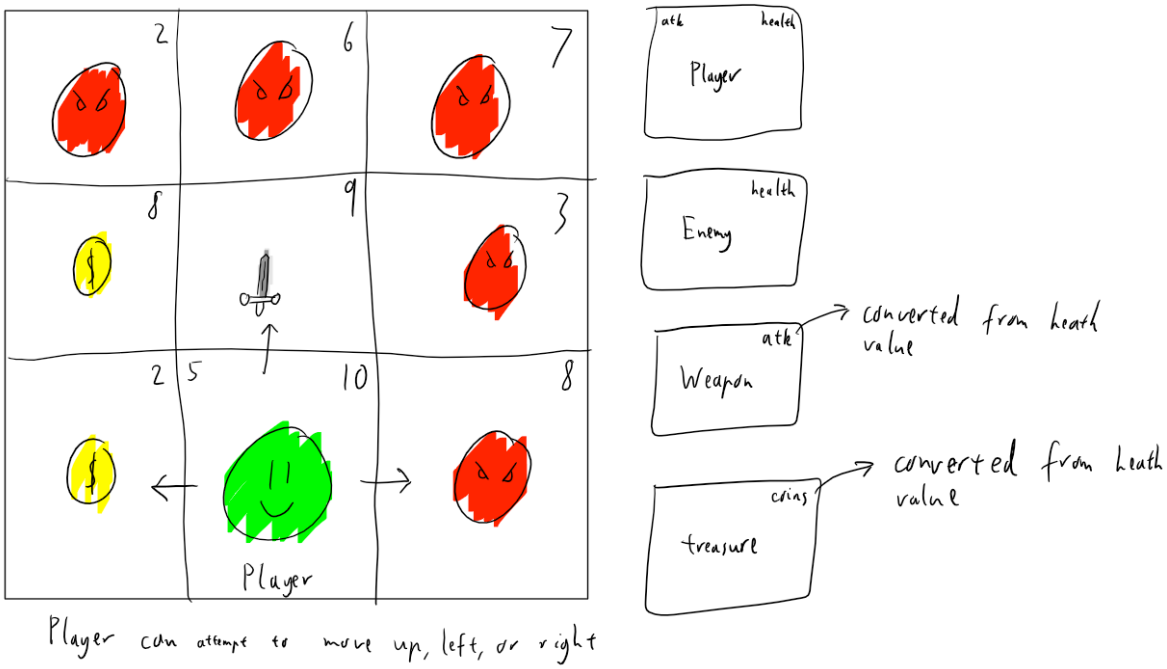
For HD, I plan to implement the factory method, strategy and singleton design patterns:

- There should only ever one dungeon at any one time, so it should be implemented as a singleton
- The dungeon would need to make new entities to fill empty dungeon cells, so it makes sense to assign that responsibility to a factory method.
- The strategy design pattern may be used to decide how the game reacts to a player attempting to move into cells occupied by different entities.

Here is how I will attempt to ensure my program will be complex enough for HD:

- Implementation of multiple game states such as a title screen and a game end screen
- Being able to start a new game after one has been created
- Introduce a high-score system (requires the use of files)
- Ensure the program is scalable through the use of the entity, cell and strategy systems
- Allow for general classes such as the entity class to be able to be reused in other projects

Include a sketch of sample output to illustrate your idea.



Required Roles

Describe each of the classes, interfaces, and any enumerations you will create. Use a different table to describe each role you will have, using the following table templates.

Table 1: Entity class details

Responsibility	Type Details	Notes
Entity class	Health: integer Methods: Draw	Abstract

Table 2: Player class details

Responsibility	Type Details	Notes
Player entity	Health: integer Attack: integer Methods: Draw	Inherits from entity class

Table 3: Treasure class details

Responsibility	Type Details	Notes
Treasure entity	Health: integer Methods:	Inherits from entity class Health is converted into coin value

	Draw	

Table 4: Weapon class details

Responsibility	Type Details	Notes
Weapon entity	Health: integer Methods: Draw	Inherits from entity class Health is converted into attack value

Table 5: Enemy class details

Responsibility	Type Details	Notes
Enemy entity	Health: integer Methods: Draw	Inherits from entity class

Table 6: Cell class details

Responsibility	Type Details	Notes
Dungeon cell	Type of cell: Entity X: column of cell Y: row of cell Methods: Draw cell Clear cell	Multiple of these make up the dungeon.

Table 7: Dungeon class details

Responsibility	Type Details	Notes
Dungeon manager (dungeon master)	Dungeon grid: Cell[y][x] Player score: integer Methods: Get Instance Draw Dungeon Reset dungeon Move entity Shift dungeon Fill empty cells	Only one of these

Table 8: Dungeon class details

Responsibility	Type Details	Notes
Entity factory	Methods: Create entity	Factory method design pattern

Table 9: Strategy interface details

Responsibility	Type Details	Notes
Handle game reaction to player movement	Methods: Execute	Strategies will be based off this interface

Table 10: Entity type details

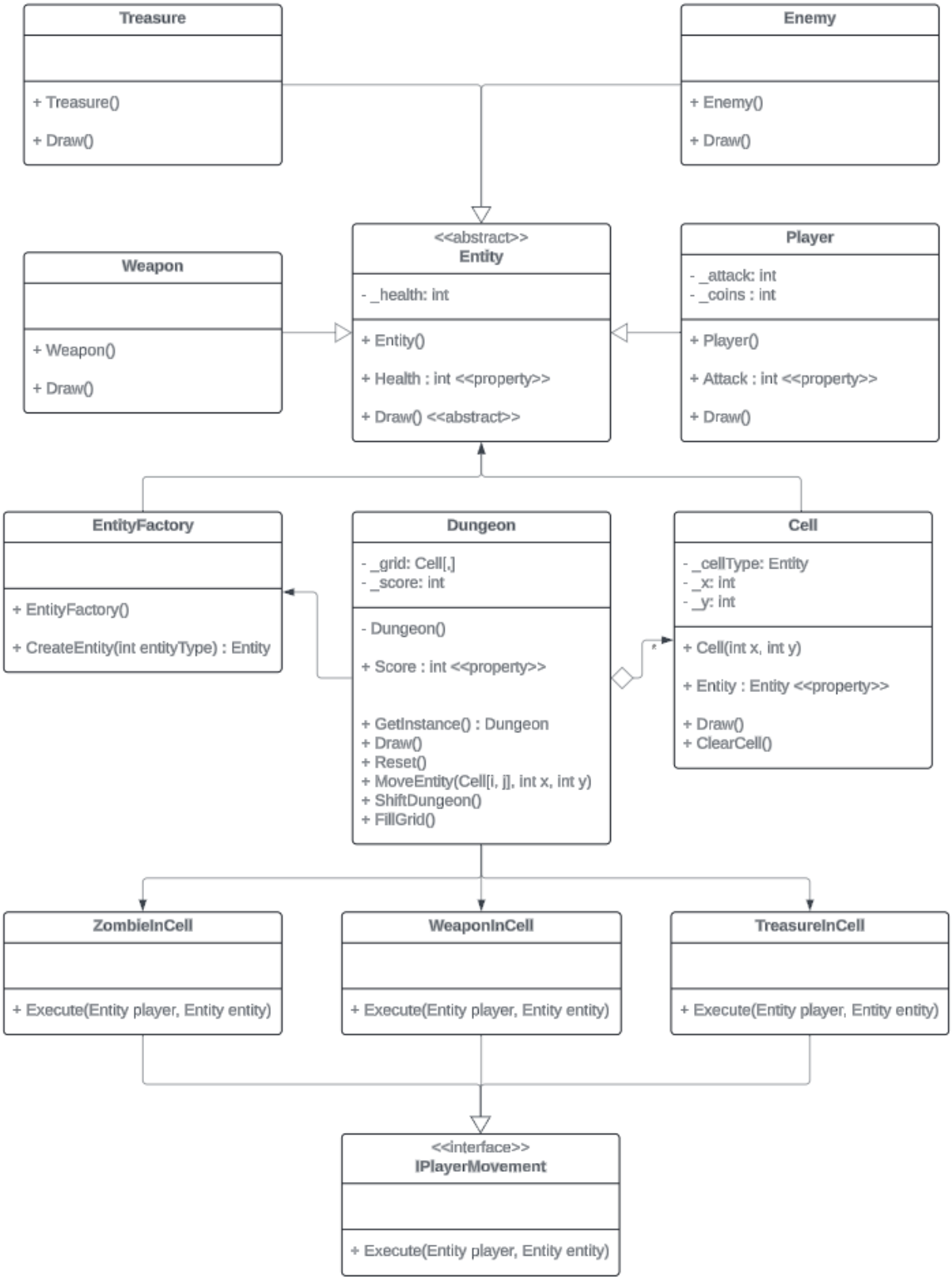
Value	Notes
0: Empty 1: Player 2: Monster 3: Treasure 4: Weapon	Used for determining the type of entity (might change).

Table 11: Gamestate details

Value	Notes
0: Empty 1: Title 2: PlayerTurn 3: DungeonTurn 4: Death	Used for determining the current state of the game (might change).

Class Diagram

Provide an initial design for your program in the form of a class diagram.
There is more that I will add (such as a factory), since I am trying to get a HD. This is the bare minimum of what would be required to have a functional game for now:



Sequence Diagram

Provide a sequence diagram showing how your proposed classes will interact to achieve a specific piece of functionality in your program.

Moving a player up (Everything in the switch will be allocated to strategy objects, but otherwise this sequence diagram stays relatively the same):