

OOP Exam

Proficiency Assessment Test – Architecture Documentation

Mohamed Serry

sTUDENT  DUBAI, UAE

Object Oriented Programming in C++

**Q1: Implement Strategy Pattern to Encapsulate Behaviors of Tiles**

Initially, the design for the tile behaviors included duplication of code between the mimic tile, bomb tile and treasure tile. The mimic tile looked like a treasure tile but behaved like a bomb tile. After studying the strategy pattern, the architecture I came up with is the following:

Tile Interaction

Damage

Gold Collect

Tile Type

Treasure Tile

Bomb Tile

As see from the above figure, tile behaviors (interactions) have been encapsulated behind the *TileInteraction* class, each tile is composed of an interaction. Upon instantiation of the tile, it simply initializes the appropriate interaction **strategy**. Furthermore, the mimic tile is now a derived class of the bomb tile as they are identical in behavior but different in states.

**Q2: Usage of State (Is a state machine worth it?)**

No, state is not necessary since the use here is in a different context. the state enum's real value is in the Render() method. however, from a behavior point of view, each tile does 1 thing only so there is no variance in its behavior based on its state. therefore, the enum is just fine as it is. Furthermore, we used the strategy pattern to de-couple behavior from the entities (tiles) and to remove duplication. should the tiles require to have more than one behavior based on a different state, then the state pattern would be a better choice.

**Q3: Implementing a mimic detector**

For mimic detection, my initial design / thought was to allow the player to reference the world since the world class includes the tiles. This is my initial pseudo design for mimic detection:

* *Get Adjacent tiles to player*
* *Loop over adjacent tiles*
  + *If the type () is mimic*
  + *Set state as revealed.*

Initially, I struggled with figuring out a place for the mimic detection method(), if it’s in the player that means the player class will access and modify tiles which I don’t think is a good way of going about things. Below is my function flow (function call hierarchy) diagram showing the architecture. Perhaps when I start refactoring I’ll get back to this and sort it in a better way.

**World.Update() -> InputHandler.Update() -> if (Key Press = E) -> GetPlayer() -> Detect Mimics() -> GetWorld() -> GetAdjacentTiles() -> SetSate (Revealed).**

Now there is many places with circular references, it seems the world knows about the tiles and it can set them to revealed. My current problem is if DetectMimics() is not on the player. It should be where?

**Q4: Adding Enemies to the game.**

**Q5: Adding Behaviors to the enemies.**

**Q8: Adding Teleporter Tiles.**

**Q6: Architectural Refactoring (As much as time allows)**

**Q7: Architectural Documentation & Decisions.**