Requirement 4

**Title:** The Treasure Room

**Description:** There is a secret key hidden on the map that Mario can only find when he is within its vicinity. This secret key will unlock the mysterious door on the map that will lead to a treasure room with magical items and money found inside.

**Explanation as to why it adheres to the SOLID Principles:**

**Single Responsibility Principle:**

Firstly, I created new classes for each game map. This allowed us to have an individual class for each map with their own single responsibility, instead of having the Application task trying to manage each map in that one file.

Secondly, the secret key class has only one responsibility and that is to be picked up by the player and used to open the door. It isn’t used in any other parts of the game and doesn’t have any other responsibilities.

I have implemented more actions that allow the game to have a bit more depth. While having opening the door and entering the door are separate actions and have their own single responsibility as such, these could be slightly too specific. But, they do only have one responsibility.

Furthermore, I have tried to implement meaningful grounds that are used in this task, including a treasure chest and a locked/open door. These are there purely for purposes of making the game a bit more fun and engaging and don’t try to manage other parts of the game.

**Open/Closed Principle:**

Like mentioned before, I added the map abstraction for this purpose entirely. Instead of modifying the application task to create new maps and perhaps new actions to move between the maps, I created the Map abstract class to give every map it’s game map and a description of what the map is called. This means that when a new map is to be implemented, it can simply be inherited from Map and it has all the necessary details to fit into our game.

**Liskov Substitution Principle:**

All inherited code that I have made throws an error if it does not implement all abstract methods that the parent class or interface can perform. I’m not sure if this is a Java thing or a thing with my IDE, but it makes following this principle trivial. While I didn’t implement too much abstraction into my game, each action has an execute and a menu description and is inherited from Action.

**Interface Segregation Principle:**

I have not used any interfaces in this design, so this principle is adhered to with this regard.

**Dependency Inversion Principle:**

Again, the Map abstraction that I have implemented allows us to easily adhere to this principle not only in this task but in the first task with the lava map, if that needs any more changing. However, there has been no use of an interface for this task which limits my discussion in this regard.

|  |  |
| --- | --- |
| **Requirements** | **Features (HOW) / Your Approach/ Answer** |
| **Must use at least 2 classes from the engine package**  Uses Action, Ground, GameMap and Item | **Key:** The key is an item that is inherited from Item. I have placed it in a certain corner of the map to make it harder for the player to find.  The fog of war aspect with the key is implemented in the Player class and if the player walks within 5 blocks of where the key lies, I call the setGround method to add the key to the map. This makes this easier than having a flag to determine whether to hide the key or not. It does however, mean that we repeatedly place new instances of the key onto the ground instead of hiding and showing one single instance. I tried to implement it that way, but had some difficulties. I chose to do this in the Player class because I couldn’t access the player’s location in the item’s class. Other than this, the key acts the same was as any item. It can be picked up and dropped and when the player has the key, they are granted with a Status called HAS\_KEY. |
| **Must use/reuse at least one existing feature**  Uses the magical items and coins that we implemented in Assignment 2, as well as various actions that need to be used to get to the key. | **Treasure Room:** If the player has the key, they are granted with the status as mentioned before. This status means that the player has access to the action to unlock the locked door on the map. I wanted to add an unlock and enter action because it makes the game seem a little more realistic with a bit more involvement from the player. I simply could’ve added one action which allows the player to enter the door but adding an unlock action which changes the door I think makes the game slightly cooler. Once the door is unlocked (OpenDoor is placed) the player has an action to enter the door. This action is similar to teleport, except it is only used at the open doors and just takes the player from the home map to the treasure room. This uses the moveActor method to move them between maps. Once the player is in the treasure room they can exit by entering the door or walk towards the chest. Just like the door, I made an open chest action and a takeitem from chest action. This is because it adds a bit of mystery and a bit more involvement to the game. The take item is added because we are taking an item from the chest, not off the ground, hence a pick up action wouldn’t work. And since we aren’t buying the items, the buyaction wouldn’t suffice either. The chest has an inventory and for every item in the inventory, the player can take it for free. This allows us to add more items to the chest or remove items from the chest easily without having implications on the other code.  When the treasure map is initialised, the coins and grounds are added afterwards to avoid modifying the groundfactory initialisation. |
| **Must use existing or create new abstractions**  Created a new Map abstraction and uses abstractions with Ground, Item and Action | This task was added to add a bit of mystery to the game and to benefit the player from exploring the map. This also should give the player a bit more of a chance to defeat bowser and other hard enemies. |
| **Must use or create new capabilities**  Created a new capability called HAS\_KEY to determine whether the player has the key to enter the door. (This saves us from creating a lot of trivial actions).  Additionally, I used the HOSTILE\_TO\_ENEMY capability to help us determine which grounds the only the player can enter and enemies cannot. |  |