Team 82 – Seng 201 Project Report

Ngoc Le – *ID: 46909266*

Anh Le – *ID: 51227573*

All the class files are split into three main categories. GUI, models, and services. To help with designing the code structure, we started by creating a class diagram. For the services class, we have EnvironmentManager, TowerService, ShopService and InventoryService. For models we created Tower, Cart, and PurchasableItem class. Our preliminary diagram is shown below.

Our EnvironmentManager class holds all the necessary game variables. Examples include the current round number, game and round difficulty, and the amount of points the player has earned. We wanted to track all these important variables inside one class to keep them in one place. The TowerService class interfaces with the InventoryController GUI which controls the inventory screen where you can apply upgrades to your towers. TowerService contains helper methods to upgrade a tower's stats, like decreasing the recovery time, or upgrading the resource amount that is shot into the cart each time.

The InventoryService class contains methods and variables related to the shop and its GUI. This class interfaces with its corresponding GUI class, InventoryController. It contains methods including setPlayerCoins, sellTower, and upgradeTower.

Initially, we created RoundManager class to contains methods for running RoundGame Controller per difficulty levels with properties such as track distance . However, we decided to remove it since we applied TranslateTransition class in Java, which already provides basic functionality defined in Animation for cart running different game modes. The Easy, Moderate and Challenging GameController will share startAnimation methods, inputs and get different cart's property stats from Cart class.

We have the UpgradeItems and Tower are inherited from PurchasableItem, which shared the getName() and getCost()

For our tests, with CartServiceTest, TowerServiceTest, InventoryServiceTest and ShopServiceTest, we tested methods that are not getter and setter, except the RandomEventService class. The RandomEventService class is a special case as it does the job for its purpose with strict method conditions but a bit tricky for unit testing because we cannot pass the random number unless it is called during implementation. GUI package is involved to animation so we cannot test these classes.

For the project feedback:

Things went well	Things did not go well	Improvements
• We had pretty good early start from the break to outline the overall game design and assign tasks for each other. • Team communication is good from the start and well maintain till the end of project • We both invest a good amount of time to share the workload in both university computer lab and at home. That is why we both have 2 different git user names each (Mickey Le – Ngoc Le and Anhle65 -ale155). Together, we have 231 git commits. • In the end, we have the game running pretty close to what we expected.	• Time management. We did not have much time for the last week to test as much as we wanted • For code design, we could have better well maintaining code blocks but we were running out of time. • We missed some key points in game requirements and took us a fair amount of work to adjust it.	• Better time allocated for the project and other papers. • Attention to details/requirements is a must! • Tasks assign could be more details so that both team member understand other's tasks.

• The effort spent (in hours) on the project per student.

Mickey Le (Ngoc Le): 150hrs

Anh Le: 150hrs

• A statement of agreed percentage contribution from both partner

We agreed the percentage contribution of each member are equally contributed with 50-50!