**SENG 201- Heroes and Villains**

*Vikas Shenoy - 53329413*
*Derrick Edward - 18017758*

The classes in the Heroes and Villains game is split between model classes and GUI classes. Model classes were made for objects such as teams, heroes, villains, cities, powerups, healing items, and maps. These classes hold core attributes for the components of the game, like the names of objects, prices of items, and special abilities such as damage reduction. Methods enabled basic operations in the game, such as taking damage, and adding heroes to the team.

Inheritance was used between parent and subclasses in multiple places. Parent classes were made for 'SuperHeroes', 'SuperVillains', 'PowerUps', and 'HealingItems'. Specific types of these objects were then created as subclasses, for example, 'PotionSmall' for 'HealingItems', 'Aether' for 'PowerUps', and 'LexLuthor' for 'SuperVillains'. These subclasses meant that multiple specific items for each parent class could be created and added in places such as the shop and power up den, in the game. A 'BlackPanther' subclass could be created if the user chose the BlackPanther hero type, and this gave access to both Black Panther's special abilities(damage reduction), and parent class methods, such as taking damage and using power ups.

The core of the game's structure is the game manager, 'Game Environment GUI'. This class manages the opening and closing of all the GUI screen classes. An instance of it is created in the main method, and this is subsequently passed screens as they are called ('SetupScreen', 'HeroSelectScreen', EnterCityScreen' … ). The game manager aids communication between classes, as it saves the user's team(which contains heroes), and cities the user will traverse(which contain villains, and shop items). This means that the game manager gives access to the GUI screen classes of all of the unit classes' attributes and methods, which is useful when performing operations such as combat, healing, and purchasing things in the shop.

The collection type used in almost all cases was the ArrayList. ArrayLists were chosen over Arrays or other collection types because they enabled the creation of collections with unknown sizes, and the easy removal of elements. One use was for combo boxes in the shop, displaying purchasable items to the user. When the user selected an item to buy, the index of the combobox selection could be looked up, and the corresponding item deleted from the shop items ArrayList, and added to the teams' item ArrayList. Arrays were also used to a lesser extent.

JUnit testing was performed on basic main classes. In the Shop class of the project, specific functions randomly generated the amount of items that can be displayed in the shop. This was executed by using a switch, with each case either adding 0, 1, 2 or all of the items. Using an assertEquals test could not be done through JUnit however manual testing ensured it was functional, hence was not included in the JUnit Testing file. Testing graphical screens was unnecessary, as most code was auto-generated through WindowBuilder, with only Action Performers requiring logical processes and calculations. Therefore the JUnit testing coverage was only 15% of the project. This was expected as only the fundamental small classes that held the attributes of the hero, villains and cities made up the majority of the JUnit tests. Additionally most bugs and logic errors are fixed through vigorous manual testing of the game, thus making the coverage a low percentage.

Numerous things went well during the creation of the game. There were not many difficulties encountered while coding. Planning at the beginning of the project led to a clear idea of how subclasses would be made for parent classes. The splitting of screens to implement gave each party clear tasks to complete. We then found and fixed bugs in each other's code. Both parties enjoyed the theme chosen for the game, and so time was managed well to finish the core functionality of the game early. Overall the game's planning and execution was done effectively with both parties producing a high quality game that was enjoyable to play. Even unbiased external opinions were gathered in order to catch anything that the game creators could not see, which further agreed to that opinion that the game works and plays well.

Early on, the project had an imbalanced contribution rate with Vikas doing majority of the early basic classes. This was primarily due to the conflicts of studying different disciplines, with assignments and test on at various times. However towards the middle and final parts of the project, contributions became much more evenly spread. Another problem was our unfamiliarity with threads, which led to difficulty with implementing a timer which would tick in the background of the game.

Contributions: Vikas 60%, Derrick 40%.
Vikas' key contributions included making model classes, city generation, combat/special ability logic, and the core structure of the game manager. Vikas set up the structure of passing the game manager to GUI screens, and enabling communication through the saving of the team/city/villain to the game manager. The city/shop random generation of locations/items, and hospital timing features were also implemented by Vikas. He implemented the setup portion of the game, including generating a cities, list, setup screens, and creating the team and its heroes. The functions for displaying team stats, item attributes, and special abilities being applied in combat were another contribution.

Derrick's Key contributions included making the logic and graphical user interface for all 3 mini battles the user plays. Derrick also created the logic and graphical interface of the hospital, lair and villain screens. Logic for the time application in the hospital was done by Vikas, however the additional Den time application was Derrick's adaptation. 80% of the JUnits Tests and 40% of the Javadoc was Derrick's completion. An overall aesthetic improvement, javadoc, and majority of game bugs were handled jointly between Derrick and Vikas.

The assignment was highly enjoyable to work on. This was due to there being minimal constraints on aspects of the game, so we were able to give it a comic book theme, which made creating specific heroes/powerups/villains easy and fun. Lab 7 on managing GUI screens was helpful in giving a template on how the program could flow. The requirements gave us a good incentive to learn about new things, such as an action listener and Swing Timer for measuring heal time in the hospital.

The command line interface section of the project wasn't particularly helpful in working towards the GUI portion of the project, and consumed a lot of time early on. We encountered a lot of problems with Git such as merging, due to our unfamiliarity with using it, and seeing as Git was basically essential for working together on the project, it would make sense to have a thorough tutorial of it earlier in the year. Overall it was an enjoyable project that gave an invaluable practical application on how large complex projects are planned and executed.