Yun Zhou 300442776

SWEN225 Project Report

Commit Table

I'm responsible for the Monkey test module.
There are 3 commits and 1st is the earliest, 3rd is the latest commit.

	Commit-URLs	**Description**
1.	9f2f88ef https://gitlab.ecs.vuw.ac.nz/course-work/swen225/2020/groupproject/team26/chipschallenge/-/commit/9f2f88efb83978588c058e54b82cb56514360d55	Frequently used movement methods and variables should not stay in the Junit Test classes as it **violates the design principle**. Therefore, I extract them into a super abstract class(Helper class) and let each Test class extend it, easy for JUnit test methods to invoke and look concise.
2.	06c8f6ec https://gitlab.ecs.vuw.ac.nz/course-work/swen225/2020/groupproject/team26/chipschallenge/-/commit/06c8f6ecc8962457f3277d9c64c9ea197e02ee5a	The **Strategy pattern** for different movement strategies **is used**, which has the interface-based programming principle. The code is elegant and extensible, refuse to add more movement methods in Helper Class in the future, which close for modifications(e.g.add movement methods) and open for extensions (e.g. implements the MovementStratgies).
3.	1bc81083 https://gitlab.ecs.vuw.ac.nz/course-work/swen225/2020/groupproject/team26/chipschallenge/-/commit/1bc81083068b93ed5b037943cf321fc701519f16	For this commit, all three movement strategies are enabled and be used, the Context class is responsible for using and swapping different strategies to let the Client classes (i.e. Helper, HandlessMaze, MazeTest etc) access and use. As we can see that in this commit, a big JUnit Test is generated and it use all three strategies through the context for finding exceptions/errors.

Yun Zhou 300442776

Ranking

A.	Oscar Sykes	sykesosca@vuw.ac.nz	5
B.	Cecilia Lu (Qing Lu)	luqing@myvuw.ac.nz	5
C.	Daniel Marshall	marshadani@ecs.vuw.ac.nz	5
D.	Samuel (Felix) Kain	kainsamu@myvuw.ac.nz	5
E.	Ruiyang Zhang	zhangruiy@ecs.vuw.ac.nz	5
F.	Yun Zhou(Me)	zhouyun@myvuw.ac.nz	3.5

Reflection

What knowledge and/or experience have I gained?

For the Cluedo project before, the code part was updated by the individual via the social media times by times, everyone needed to take lots of time to figure out where to implement a new feature and debug. Except this, **only** one person could update the project code at the same time because we could not ensure that there will be no conflicts, **thus, the development for Cluedo was not efficiency and not convenient.**

To better connect with the world and conduct collaborative development more efficiently, the distributed version control system Git is used by us for this project. Unlike the Cluedo project before, git allows for the parallel development where it will merge the pushed code from team members automatically if there is no conflict within the same class. Except this, git also allows us to revert to the history version, easy for us to compare and debug.

For instance, the pushed new version can not run due to something really important has been deleted accidentally. Within use git, we can revert to the unaffected version and compare it to the new version in order to find which part has been deleted, **we can see that Git is pretty convenient to use and much more efficient during development, the experience of using Git is a precious to me in this project.**

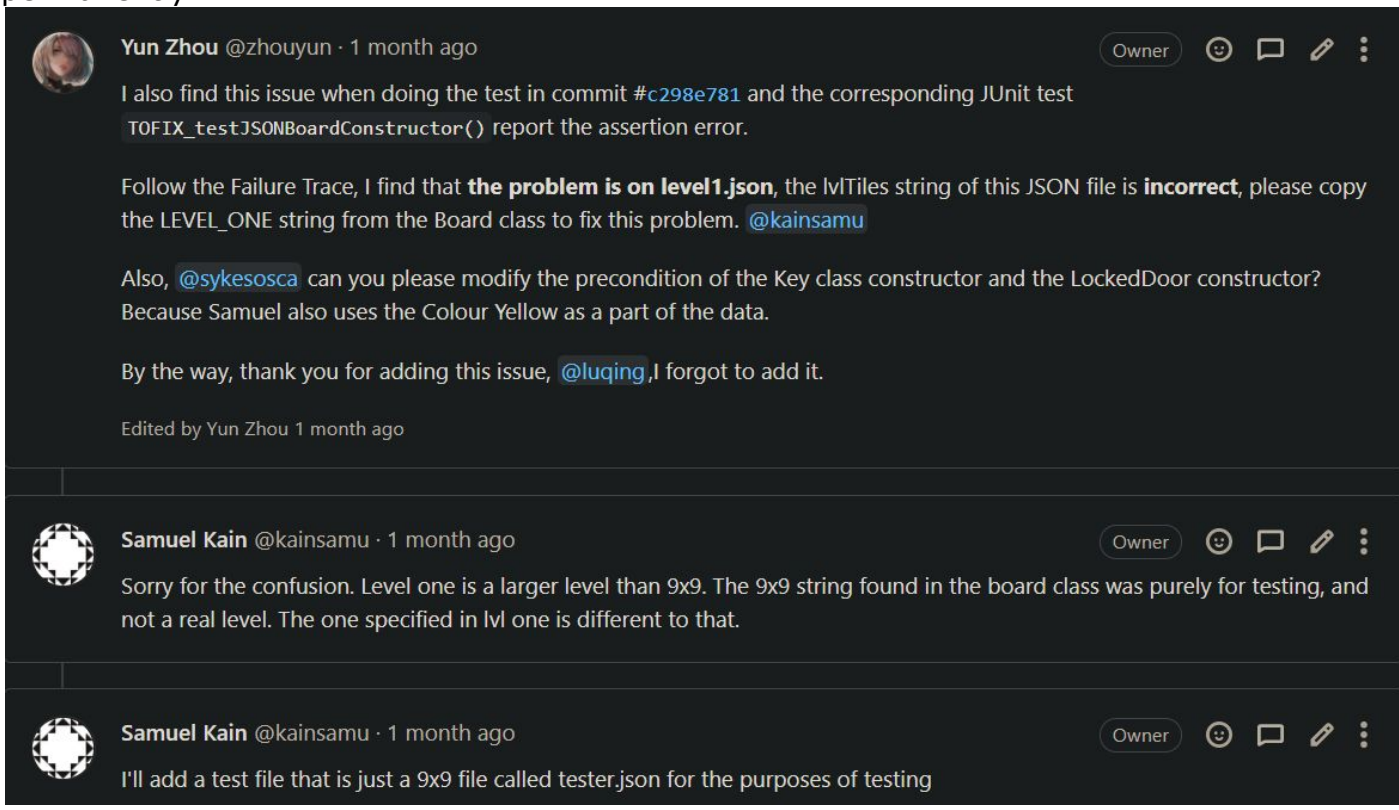
Except for the Git, I also **gain knowledge of the design principle**. From slides, I obtain that a good design should be **high cohesion and low coupling** which is the stuff that every programmer want to achieve(at least I am). This project is a typical practice of the design principle, **the dependencies restrict the relationship between each module furthest so that it's low coupling which means they are independent**. For instance, when the modification is needed in the future in order for running the game on a mobile device, we only need to change the Application and Renderer modules, other modules like maze and persistence modules are not affected as they are not responsible for adapting mobile stuff. **The high cohesion and low coupling design are pretty easy for us to maintain and implement a new feature.**

In conclusion, I gain the experience of using Git which is as a team how to develop the project more efficient, also, I gain the knowledge of the design principle which helps us to write a good and maintainable project. Except this, writing tests help me to understand how the project works, what structure can be optimized.

Yun Zhou 300442776

What were the major challenges, and how did we solve them?

In my opinion, the communication and debug the project are major challenges. It's not surprising that bugs are found by the team member who is responsible for other modules during the development. Generally, the bug should be fixed as soon as it's been found, but due to the privilege has been restricted, the bug must be fixed by the member who is responsible for that. For solving this, it not only requires good communication but also these issues must be recorded for members to manage and solve easily.
We overcome these challenges by using the GitLab Issue tracking system. Within use the GitLab, when someone finds an issue, the description of the issue will be recorded, described and assigned to the specified team member.
For instance, before the integration day, there was an issue (i.e. issue 20) updated by the member who was responsible for the Application module and it pointed out that the new Board constructor base on the Persistence Module could not be used which proved that the issue should be somewhere in Maze or/and Persistence module. **As the member who was responsible for the monkey module, through the test, I successfully found that the issue was the JSON file that represented the level was wrong so that the Precondition check on Maze.Board constructor refuse and throw the exception, so, I wrote the comment under this Issue 20 and assign it to my team members, finally within use the GitLab Issue tracking system, we were well communicated and solved this issue together successfully**, and then this Issue was closed by us on GitLab so that we knew that this issue was solved permanently.



Yun Zhou 300442776

Which technologies and methods worked for me and the team, and which didn't, and why?

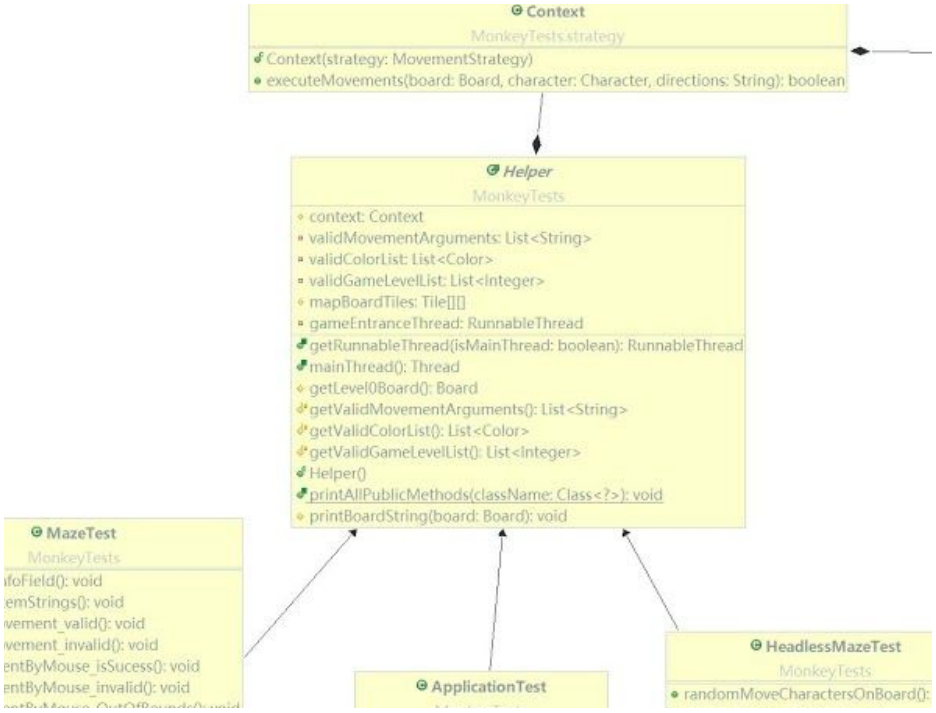
- The GitLab worked for all team members, really helpful. Within use the GitLab issue tracking system, projects were planned, tasks were assigned to the specified member also it helped us to manage bugs.
- Facebook was used by us to organize meetings weekly in order for discussing and communicating about the project, such as what to do next, what to fix, where to implement the new feature, what's the class that be used to interact with other modules, etc.
- The Google Precondition API was another technology that works for me and the team. This precondition API can not only let the code concise but also simplify the if-else check so that it improved the efficiency, really helpful.
- The plugin SpotBugs worked for all team members. This plugin helped us to reduce and subtract any potential exceptions/errors, really helpful.
- IDE Reference Browser helped us to figure out whether the module dependencies meet the requirements or not. The figure was pretty straight forward, any unexpected dependencies could be seen from this in order for locating the issue quickly.
- EMMA tool that Eclipse has worked for me because by using this I knew how many remaining that I should test.
- We did not use the UML due to we didn't have time to figure out how to use the umldocklet, this technology should be used next time.

Discuss how you used one particular design pattern or code contract in your module. What were the pros and cons of using the design pattern or contract in the context of this project?

The Monkey Test module was my duty which needs to detect violations of the game logic which need to find a general programming error or postcondition or invariant violation.

1. In the beginning, Junit test classes were only classes in my package and lots of variables/methods were a part of these classes.
2. Then, I realized that it violated the design principle which violated the single responsibility Principle, the cohesion was too low. Therefore, for optimizing the design, I extracted *them* into a super abstract class(Helper class) and let each Test class extend it, easy for JUnit test methods to invoke and look concise so that each test class only responsible for testing.(9f2f88ef)

3. I still not satisfied with the Helper class as the cohesion was still low, the Helper class was still responsible for too many things. Due to the movement of Characters (i.e Chip and Bug) was the core component of the game and it required different types of movement strategies, so I extracted different movement strategies in Helper class again and enabled to use the Strategy pattern for them which you can see the diagram above. (06c8f6ec)
4. After these refactorings, finally, each class got lightweight responsibilities in which the Strategy objects represent different types of strategies and the Context class whose behaviour changes as the strategy object changes, the behaviour depends on the type of the Strategy object, the Helper class is responsible for storing all variables that needed for testing and using, easy for test classes to access and invoke.



➢ **Pros:**

- The responsibility of each class in this strategy pattern is clear, the code is concise, elegant and clean.
- Different strategies can be swapped easily by utilizing the Context class.
- The format of the strategy methods is fixed which means arguments are fixed, so unnecessary redundant multiple condition judgments are avoided.
- It satisfies the design principles. It has the interface-based programming principle, the code is **extensible**, refuse to add more movement methods in Helper Class and Test Classes, which close for modifications(e.g.add more movement methods) and open for extensions (e.g. implements new MovementStratgies such as AIMovementStrategy).

- **Cons**:
    - There will be more strategy classes which mean strategy inflation. That is, strategies may not be used frequently thus brings inconvenience and inflation.
    - All strategies are exposed to the outside which means not very secure.

I think that if the strategies are more than 5, then we should consider mixing other patterns to fix the problem of strategy inflation. In my opinion, the Decorator pattern that belongs to the Structural pattern may be used to decorate which means enhance the strategy, such as the RandomMoveStrategyBug and AIMovementStratgy(not implemented) can be enhanced so that movements on Bugs can be merged and not the individual strategy, which fixes the strategy inflation issue.

What would I do differently if I had to do this project again?

As we can see from the previous question, I enable the Helper class as the bridge in which every JUnit Test classes extends and use it directly. The Helper class contains every data that need to use frequently, such as the Context variable to swap the strategy, valid game level list and other valid arguments lists etc. It is my **self-made class that only contains the data,** but it got some problem and can do some optimizations, which is:

- In context class, the setter method should be constructed, so that when swapping the strategy, I just need to use context.setStrategy(Strategy), don't need to use context = new Context(Strategy) again and again, which make it more concise.
- lists use the lazy initialisation in which it initializes the list in the getter method, it adds a side-effect to the getter, should use the official data class from Maze package.

Except for the bullet points above, there are still some points that I will do if I can do the project again, which is :

- allocate time reasonably, due to the personal issue(I attend a funeral) and assignments from other courses, I was busy and not have too many time to do my parts for a long time, I think I could allocate the time better.
- communicate with my teammates more closely to collaborate the project more efficient, I should communicate with members who responsible for Maze and Application more closely in order for debugging the project and know what kinds of tests do we need from each point.
- Gitlab issue tracking system should be used more for planing the module and assigning the task.
- UML should be used to plan and include in the project.
- Try to optimize the Strategy so that the Random movement on Bug can be merged.
- Try to implement the new feature of the Strategy such as the AI movement strategy.

All the bullet points of this question above are the things that I want to optimize if I have to do this project again.

What should the team do differently if we had to do this project again?

Our team is well-organised and friendly, but we still have some problems in which:

- Everyone takes different courses so that everyone is busy at a different time, there should be more meetings to discuss and plan the project such as what interface can be used to interact with the specified module.
- We did not generate the UML diagram of our module so that we could not see the structure of our package directly. it's not good because the UML is a visible diagram that can help us to figure out the relationship between each class for adjusting and applying the design pattern.
- The Application module should try to use the design pattern, the cohesion was too low on the Application Module, the GUI class was responsible for too many things such as interact with all modules, everything was added into GUI class. It violated the design principle which is the single responsibility principle, interface-based programming and closes for modification, open for extensions.
    - Should try to enable the MVC pattern, the MVC pattern can separate the layer so that the cohesion can be higher, the Model class from MVC can also be used by me to test. (I cannot test for the Application because everything is embedded into the GUI constructor, there are no methods that I can utilize to test.)
- We should enable other branches such as the feature branch to implement the new feature, the develop branch which is the super branch of feature branch that contains the code that has feature without any bugs. Also, the hotfix branch to fix any bugs in emergencies that based upon the Master branch. The Master branch should always be stable which is consist of the develop branch and hotfix branch.
- The movement strategy on Bug can be better, from now on, the Bug can only move when the Chip is moved, can let the Bug move regularly.
- The AI movement strategy can be enabled so that the Chip can play the game automatically and different Bugs can have different methods to move around the Board, such as regular movement, when the chip is close enough then add the velocity to try to catch the Chip, etc.

Conclusions:

In conclusion, I have gained a lot according to this course, it reminds me how important the design it is, a good design can not only help us to save the cost but also improve the efficiency. CRC cards record the responsibilities of each class so that I can know what to do next, the UML diagram makes the structure of the project visible, the design pattern improves the quality of the code as well as it saves the total cost. Also, experiences of using git and work as a team are precious that help us to adapt to the future.