CMPT 276: Group Project
Phase 2
Group 3: Ethan Rowat, Luna Sang, Pavel Jordanov, Tai chuan david Png

1. Overall Approach

Our overall approach to implementing the game involved a variety of methods including collaboration, communication, research, restructuring, and simplifying our design in order to make our implementation better. At first, we started with our UML diagram and built the classes according to the specifications. We then reassessed the classes to find what was missing to build a working software. As a result, many of the UML classes were restructured to fit the current design. The entire group met several times to share ideas and find ways to meet the requirements of the project. We searched online for possible design patterns that would offer solutions to our problems. It was found that our design had some methods that had to be placed elsewhere in our project. We researched a GUI library and decided on JavaFX as it had plenty of documentation and was easy to implement. After these iterations, we began to change preliminary classes with more detailed classes for what was required. Furthermore, we implemented necessary features for the game. Features often were implemented on a developer's branch and then merged with master once the feature was complete. Our team created richer use case implementations once we observed our results. Difficult features were evenly split among the developers. There was ample collaboration when dealing with merge conflicts, git operations, implementation and interface of code.

2. Adjustments, Modifications, and Justification

Package name	Class name	Adjustments & Modifications
characters	MovingChatacters	We moved this class from gameMechenics to characters as the 2 characters' classes inherit from this class. This class is not abstract in our implementation because we found that the functions in this class can be reused by the child classes and do not need to re-implement in each child class.
	<<Interface>> Movable	We put the functionality of the moveable interface into the MovingCharacter class as the Movable interface is only used by the MovingCharacter, so we merged the Movable interface with the MovingCharacter superclass.
	Wilbur	- We added more pictures for Wilbur when moving in different directions in order to make it more vivid when moving. - An initPlayerImages and a draw function were added to achieve the changing image.

	Butcher	- We changed the name of the attack function into tracking as this function is used for tracking Wilbur and is more intuitive. - We added attributes of probabilities which are used to alter the butchers' movement. So a butcher can move randomly in a probability level set by the probability attributes.
gamemechenics	Path and Barriers	We moved the Path and Barrieres classes from the environment package to the gameMechenics package because they extend BoardSpace.
	Board	- We used a JSON file to import all the coordinates of game elements. - Our draw and paintCharacter methods of all classes were also moved to the Board class since that was the class with the GUI creation and redrawing. However the setting of images was implemented in each respective class and done during creation of the object. - The EventRunner class and KeyListener interface was implemented as an interface and series of fields within the board.
	BoardSpace	- We changed the BoardSpace array that would hold board spaces into an integer array, this made it easier to represent what was lying on each board space and would allow us to quickly and efficiently check the positions on the board. Along with that we also needed to add a series of values to the BoardSpace class in order to accommodate the integer array in the Board. - The isBlock function was moved to the Coordinate class since the Coordinate is in each character's class and the characters need to check for barriers and walls when moving.
	Coordinate	- We added up, down, right, left functions that are used when moving the characters. - We added checkCoordUp/Down/Right/Left functions that are used when moving characters, and this check will ensure there are no barriers or walls. This coordinate class also implements Comparable interface that is used when comparing to coordinates.
environment	BearTrap	For the concrete entities, our UML reflected that each entity would have an image file attribute, this field was extracted into their parent class for better readability and conciseness. Final values that alter points were added to StationaryModifier to make it easier to alter the values.
	MagicTruffle	
	MagicCorn	
	StationaryModifier	

3. Management process and Division of roles and Responsibilities

Throughout the game development, each group member was assigned a role. Although group members often overlapped to help each other when someone would hit a dead end.

- ★ **Ethan Rowat:** Board class, grid system, Board Space class, character movement and interaction, Coordinate class, Barrier class, Path class, merge conflicts, JSON file reading, level creation, death screen, exit screen.
- ★ **Luna Sang:** Board class, Coordinate class, Butcher class, Tracking Method, Moving Character class, Butcher movement.
- ★ **Pavel Jordanov:** Board class, Wilbur class, Tracking Method, Moving Character class, Wilbur design (This included pixelating the pig images on photoshop), Wilbur movement, menu screen.
- ★ **Tai Chuan David Png:** Board class, Magic Corn class, Magic truffle class, Bear trap class, stationary modifier class, maze design.

4. External Libraries and reasons

When researching the external library to use for the GUI, we considered a variety of options. We opted to use JavaFX. JavaFX has ample documentation and is very extensive. There were many classes that worked well for our idea of implementation, for example GridPane offered the main functionality for what is displayed to the user. JavaFX can also be added into our project via maven dependency quite easily. We also decided to use files to build the level in the code, that way level files could be switched out to change the level instead of writing code for every new level. In order to do this we needed a very simple JSON parser. We opted to use Google Code Simple JSON. This library worked perfectly since as the name states, it is simple, and it only took a maven dependency to work from it right away with its parsing ability.

5. Efforts to enhance code quality

Code reviews: Features were often implemented with a rough idea of how they were going to be completed. However in practice, some changes have to be made to the plan in order to accommodate unforeseen, arising, and unexpected behaviour. To counteract these issues, we

often reviewed code side by side in person to investigate why code did not work, how it could be improved or how the work could be lessened. This leads to refactoring of less than ideal code.

Discussing implementation: Sometimes in development there were issues that were not immediately understood. Instead of charging ahead and beginning to code (which can lead to more errors) whenever there was a difficult problem, our team would discuss how we could best solve it. Once we figured out how to solve the issue, whoever was most comfortable was assigned the task of coding the solution.

Bug checks, testing: Errors, bugs, and crashes were certainly an uncomfortable part of our progress. We worked to reduce these by attempting to make sure that the master branch always compiled and ran. This means merging and running a copy before pushing it to the main branch. Additionally, testing out edge cases, and expected features was a good way to look for bugs. For example: "Is there any code that could lead to an infinite loop?", "What will happen to the character when they reach the end of the board?", "Does the invincibility protect the character from the traps and moving enemies?". All of these tests, along with using good practices and prioritising working code helped enhance our code quality.

6. Biggest challenges during the phase

➔ Butcher's tracking function

The tracking/chase method was originally implemented using conditional statements that would check where Wilbur was on the board relative to the butcher. If Wilbur was to the left of the butcher, and there were no obstructions on the butcher's left side, then the butcher would move to the left. This same concept was applied for all directions. One of the major issues was that the butchers would catch up to Wilbur and kill him too quickly. To find a solution to this, Pavel played several rounds of Pac-man to analyse the movements and reactions of the ghosts. Pavel noticed that the ghosts would only move in the direction of Pac-man about one in every four moves, the rest were random movements. This meant that there was some sort of probability aspect involved which determined if the ghosts could "sense" where Pac-man was and move towards him. After noticing this, a probability aspect was implemented so that the butchers would only move towards Wilbur 30% of the time, and the rest would be random moves. The last major problem was that we needed to make the tracking algorithm run on the timer of the game rather than being

dependent on the movement of Wilbur. We fixed this by changing the function call to be in the same place we regulate the timer, instead of calling the function after Wilbur moves.

➔ **Git merge**

The merging of our respective branches with master once a feature had been sufficiently implemented caused a lot of issues. Especially when a feature was particularly large, or commits were not decently spaced. There were also times when an overlap of responsibility had occurred so the merge was a series of choices of which code to implement. This caused several hours of issues and even some bugs along the way. This contributed to degradation of our code and readability. The merges were difficult but were made easier when Fortunately, our code enhancement efforts helped with reducing these effects.