Scrum Approach

With the development of our game, we decided on taking more of a Scrum approach to development. Our development for this phase first began with gathering all the information possible for the completion of our project and the tasks involved with it. Every group member had a vision about the game and we made sure our “customers/stakeholders” and user-stories/use-cases were well understood and considered.

We first created our product backlog, filled with all of the features and requirements needed for us to complete the game. The product backlog was sorted in terms of priority with the item with the greatest priority being at the bottom of the product backlog. This list allowed for all group members to have an idea of what the project requirements are and which ones have the most priority/importance. Additionally, we also assigned a Story point or difficulty level for each user-story/use-case on our product backlog (where 1 is easy to 5 being difficult).

Use Case/User story	Difficulty/Story point(s) (1-5)	Priority (1-5)
As a player/inmate, I am able to get game instructions if I needed help with how to play the game	3	1
As a player, if I am in the in-game menu, I can resume my gameplay	2	2
As a player, if I am in the in-game menu, I can return to main menu	2	2
As a player/inmate, if the overall score is 0 I will lose the game	3	3
As a player/inmate, if the timer runs out I will lose the game	3	3
As a player, I can exit the game whenever I want	2	3
As a player, I can return to main menu and quit the game	2	3
As a player I can start a new game	2	3
As a player, I can save the game and continue from the level I left off	3	3
As a inmate/player, I can collide with static objects that force me to stop	3	4
As a inmate/player, I can collide with traps that decrease my overall score	4	4
As an player/inmate, I can collide with a chicken drumstick (collectible) to increase my score	4	4
As a player/inmate, I can collide with a clock (collectible) to increase the in-game timer	4	4
As a player/inmate, I can collide with a key (collectible) to increase key count and overall game score	4	4
As a player/inamte, if I collected all 10 keys, I can exit the gate and win the game/round	4	5
As a player/inmate, I can move in whatever direction I desire	3	5
As a inmate/player, I can collide with guards (moving enemies) that end that game for me	4	5

*****Note throughout the course of this phase we kept adding more user stories/use cases to the product backlog *****

With our Sprint execution, we made sure to evaluate each other’s code by using a peer evaluation method where one group member’s progress on a particular task is reviewed and accessed by another group member. Normally near the end of our current Sprint, we all get together to do a sprint review where we tend to do one final assessment of the progress we made for the sprint. We met every day to assess each other’s code and work collectively; however, every 3-4 days we have 15-minute meetings where we come together to decide on the requirements and tasks involved for every upcoming sprint and to create our next Sprint backlog.

Collectively we all acted as product owners (PO) for every upcoming sprint, collectively coming up with features and tasks that need to be completed. The development team consisted of all group members, where we worked together to develop and deliver each increment in the game's development. We collectively agreed to deliver a complete version of the current Sprint every 3-4 days. Although for every sprint we tried our best to complete them within 3-4 days there were certain sprints where we managed to complete within 1-2 days and some others within a week's time, but we set the goal to get every Sprint finished within 3-4 days.

Sprint 1

Tasks: The first Sprint consisted of implementing the game window, setting up object sprites (like the inmate, guard, collectibles, and traps), and libraries.

Difficulties/Solutions: The difficult part with this phase was familiarizing everyone with using Git, doing research for different libraries we would use, and setting up a project on Maven. Essentially this Sprint was for us to plan out the areas of our design that we did not think about in Phase 1 of our project. This Sprint involved a lot of communication and planning with group members. The daily Sprint meetings tended to last longer because we had to make sure all of our team members were comfortable using Java, git, and the IDE Intellij prior to writing any code. Once we got past that stage we first looked into how to create a Maven project using Intellij. We then started putting together all sprites we needed for our game and put them in a package in an organized fashion. Afterward, we started looking into what libraries we could use for our game implementation.

As a group we decided to use the following Java standard libraries:

- ➔ **java.lang**
- ➔ **javax.swing**
- ➔ **java.awt**
- ➔ **java.io**
- ➔ **javax.imageio**
- ➔ **java.util**
- ➔ **javax.sound**
- ➔ **java.net**
- ➔ **Java.text**

***Note throughout the course of this phase we kept adding more libraries to this list ***

We decided to use the Java standard libraries because they are very easy to use. Most frequently we use the swing and java.awt libraries for our GUI components. The java.util allowed us to use a variety of different data structures and the java.io library allowed us to use the files and images within our implementation. Furthermore, the java.lang allowed us to use Strings, enums, and doubles.

Completion time for current Sprint: 3 days

Sprint 2

Tasks: In the second Sprint we implemented level designs, inmate and guard movement, static collisions

Difficulties/Solutions: The guards' movement was difficult for us to implement because we needed to figure out a way to make the guards follow the inmate. The completion of this task was pushed to the backlogs of future Sprints. Regarding the level designs we tried to the best of our abilities to mimic our prototypes; however, they did turn out different because of the time constraints we had for the second Sprint. Furthermore, with the player movement, we started to notice different bugs with how it collided with walls and doors as the inmate sometimes partially went inside of the wall and wouldn't allow it to move at all. In terms of level design, we tried to implement different levels, at first, we hard-coded it but later on, we came up with a better design that gave us the flexibility to easily make different levels.

Completion time for current Sprint: 4 days

Sprint 3

Tasks: In the third Sprint, we implemented our collectibles (chicken drumsticks, clocks, and keys), the escape gate functionality, and the game scoreboard and timer.

Difficulties/Solutions: The completion of this Sprint was very time-consuming for the group as we encountered many issues with implementing the gate functionality. Furthermore, the implementation of the collectibles was very nuanced as every collectible is related to another system in our game. For example one of our collectibles is a clock and every time the inmate/player collects the clock, the timer increases. Many of these collectibles are related to other systems in our game and we have to make sure they are able to satisfy their purposes. Another problem we encountered was with the implementation of the escape gate, where we were faced with the problem of making sure the gate was inaccessible when the player has not collected all the keys on the map. With the collectibles, there were many intricacies and relationships that we had to make sure we were capable of working with each other.

Completion time for current Sprint: 7 days

Sprint 4

Tasks: In the fourth sprint we implemented auto-level save, menus, menu buttons, and UI options/instructions

Difficulties/Solutions:
As a group, we had a prototype we wanted to follow but with its implementation, we decided to change how the saves worked in our game. Originally we wanted the user to save the instance they left from the map and in order to do such a task they had to manually go to the in-game menu and click on the save game button; however, later on with the implementation we decided to simply just auto-save the current level the user is playing on as it just made more sense to all of us instead of the previous method. In terms of the implementation of the menu, this process was very straightforward for the development team as prototypes were there to give us a good representation of what it had to look like.

Completion time for current Sprint: 4 days

UML & Use-case Changes

Our UML was constantly subject to change throughout the course of our development. Most of the details within our UML stayed consistent although we ended up changing some of the class names. With our different types of menus, our UML design consisted of 5 different types that inherit from the superclass of menu all within the menus package. However, in our implementation, we decided to not include the superclass because of the differences with all the menus. Additionally, we decided to get rid of the difficulty menu because we decided to make the level increase in difficulty by default whenever the player/user finishes the round (the rounds progressively get more difficult). With our UML design, we have a package called entities that contains the classes actor, dynamic actor, static actor, rewards (collectibles). However, in our implementation, we decided to break that package apart into different packages. With the guard and inmate classes, we decided to keep them in the entities package, having both the classes inherit from the superclass Entity. We decided to do this because both the inmate and guard share similar characteristics with one another. Furthermore, with the collectibles (key, chicken drumstick, and clock) we had them inheriting from the superclass Reward in our UML; however, we decided to have them (along with the static items like the walls, traps, etc) in a package called objects. Within our 4th Sprint, we realized how much our design lacked in terms of UI-related classes. With our UML we only implemented a screen class which later on with our implementation we came to the realization of how poorly planned this aspect of our design was. With our code, we created a package called main that consists of the classes AssetSetter, CollisionManager, GamePanel, KeyManager, MouseManager, SoundManager, GameDisplay, and UtilityTool. Additionally, with our UML we got rid of the Score and Time classes and implemented their functionalities in the GameDisplay class.

Our use cases throughout this phase stayed mostly consistent. This allowed for our team to easily implement and program the various cases without severe changes and their associated repercussions. Some changes that we made include: removing the movement use case associated with each direction and replacing it with a single use case that encompasses movement in all directions. We also removed the instructions use case since we found that most users instinctively used the W, A, S and D keys to move. Furthermore, we also readjusted the number of keys to collect from 10 to 3, 4 and 5, depending upon the level, to shorten gametime.

Challenges

Initially, most challenges faced were related to getting every team member to set up the software required to proceed with this phase of the project. Also, understanding the process of building a Maven project and researching the libraries required to receive input from the user via keys and mouse buttons. Next, deciding upon the algorithm to be used for sensible and rational enemy/guard movements with respect to the location of the player/inmate was a tough task, since we had to weigh our options ranging from complex algorithms that are accurate and precise but extremely hard to implement versus simple algorithms that are less accurate and precise but easier to implement. We

decided to go with an algorithm that is easy to implement yet accurate as needed for our application. It works by comparing the relative positions of the inmate and the guard on the map. The level design was yet another challenge that was overcome through focusing on efficiency and flexibility to incorporate multiple levels into our game as opposed to hard-coding obstacles and collectibles onto each map. This helped us drastically decrease the time needed to implement the levels along with making the process flexible. Other difficulties include the mechanism of collecting collectibles along with their associated updates on the map and the game flow. For example, opening the escape gate when the desired number of keys have been collected, updating the score and timer, etc. Bug detection and debugging also posed problems along this development process.

Refactoring and Improvements

Some improvements can be made in areas such as file, folder and package organization. Objects such as the collectibles and the escape door have several features that are reminiscent of an entity-type object. Hence with some adjustments, these objects could be packaged along with the inmate and guard files. Wall collision is yet another area where improvements can be made. Currently, the walls seem to have a slightly wider effect on stopping the movement of entities than their size on the map, hence making the edges seem “chunkier” than usual. Also, the animation of the flashing door depends upon the number of keys collected at different levels. In level 2 the door appears to flash after the user collects 3 keys however it only opens after all 4 keys are collected. This could also be solved in future revisions.