Phase 2 Report

Overall Approach to Implementing the Game

One of the biggest challenges we faced is figuring out how to approach the implementation. Over many days, we had conflicts about whether to start with the game logic or the GUI implementation. Caleb wanted to start with the game logic first, while Rahim thought that starting with the GUI (Swing or JavaFX). Rahim thought that having a GUI, a visual representation, would help us with implementing the game logic because we would have a more "concrete image" in our heads of what we're implementing. Caleb thought that the GUI always depends on the "model" or the game logic. He thought that it doesn't make sense to implement a GUI without game logic. What would the GUI use as a basis for creating the board? How would board interactions work without game logic? Besmillah played devil's advocate on both sides.

In the end, we settled on the following compromise: we learned as much as possible about the Java GUI toolkit named Swing. We agreed that we will pass Swing a simple, hard-coded map in order to be able to test game logic in a visual way. Then, whenever we made changes to the game logic, we could immediately see a visual representation of it.

So Caleb created the initial structure with the "ArcadeGame", intending to implement the game logic first. Besmillah and Rahim ended up creating a new structure "team6game", from which we implemented our compromise.

Management Process, Division of Roles and Responsibilities

Once the basic structure of everything was agreed upon, we decided to divide roles and responsibilities as such:

- Since Rahim knew the most about the GUI, he was in charge of implementing the GUI.
- Since Caleb knew the most about the game logic, he knew which parts of the implementation were hardest, so he tried to section off the game logic between himself and Besmillah. He also told Rahim what the game logic would output (i.e. for the map, a String with "X" as barriers, "*" for punishment, etc.) so that Rahim can know what Swing should implement.
- Caleb would write the report.
- Jason will be responsible for making Javadoc comments.

Between Caleb and Besmillah, they created a skeleton code for the game logic. However, we again found it harder to work on this separately because the classes for the game logic are quite interconnected. It was decided that for this project phase, Caleb would help Besmillah figure out difficulties in game logic while Besmillah codes. This meant Besmillah's work is intricately connected to Caleb's.

It was also decided that Caleb would do the next phase of game logic coding (some to be done in this phase, some to be done in the next phase): making the game logic more cohesive, less coupled, more flexible, and more maintainable. For example, for the purpose of this phase, we have decided to hard-code barriers in the map (for why, see "Challenges Faced" below). The more completed version (in the next phase) will have the BoardGame more flexible. Another example is that the "symbol()" function is coupled with each GameObject. i.e. Punishment contains a symbol() function. We find it more cohesive to put this symbol() function as a display() function in the GameBoard. Caleb also maintained the structure of the code so that it could reflect a more object-oriented approach.

Biggest Challenges Faced

In terms of actual code, we found it hard to implement “Random” for our objects.

In particular, when we tried to put barriers into the board randomly, we would get maps where the Player is stuck from the moment the game starts. Another map had barriers around the exit, and so the Player can never win. Other maps put barriers around Rewards.

Random placement of Punishments and ChaserEnemy also complicated things. Specifically, sometimes these can be randomly placed right beside the starting point or end point. This bug still exists within our code and will be revised for Phase 3.

Adjustments and Modifications to the Initial Design

One of the biggest adjustments is that we had to hard code our map and board utilities. Cell and Direction are not implemented. Board, Position, and others are changed. GameFrame, GamePanel, MapLoader, and App (where Main lives) were added in as part of the GUI. The GameTimer we found didn’t warrant a class onto its own, so we scrapped it. This is the same for Barrier. We changed the name of MainCharacter to Player. We added more “objects” as superclasses. For example, MovingEnemy extends Enemy, which extends CharacterObject, which extends GameObject. Another example is BonusReward extends Reward, which extends CollectibleObject, which extends GameObject.

External Libraries Used + Justification

We decided that it would take a lot more work to code a GUI from scratch. We designed our BoardGame map to be represented by Strings (i.e. walls are represented by ‘X’, BonusRewards with ‘o’, etc.). However, we found it would take too long to turn this string into a graphical interface that has interactions with the user. This is why we decided to use Swing, a Java GUI toolkit.