Phase 2: Report for *North Pole Breakout* Project

Overall Approach to Implementing the Game

The main goal of our project was to build a game named North Pole Breakout, that includes multiple interactive elements like characters, rewards, walls, and other obstacles. The game is designed in a 2D grid format, where players navigate through different tiles, interact with various entities, and earn or lose points based on their choices and moves.

Our approach started by carefully defining the classes that would make up the core of our game. Each class represents an important part of the game and allows for clear and organized programming. For instance, we developed a Tile class that handles the different elements a player might encounter on each tile, such as walls, rewards, or traps. This Tile class also determines if a tile is "traversable" (walkable by the player) or blocked by something like a wall. The `occupy` method in the Tile class is an essential function that checks whether the tile is already occupied by a certain entity and then responds appropriately, either moving the player, deducting points, or adding rewards based on the game situation.

We used the Singleton pattern to create instances of Player and Game to insure only one instance of those classes exist at a given time.

Our plan was to build all the necessary classes first, such as Entity, Tile, Wall, Reward, enemies etc. Each class holds specific properties and functions that together create the game environment. After setting up the classes, we then connected them to bring the game to life, linking player actions with tile behaviors and making sure each interaction was smooth and consistent. By taking this approach, we were able to break down the project into manageable tasks, focusing on building each game component in stages, which allowed us to stay organized and motivated.

Adjustments and Modifications

As we moved forward from our initial designs, we realized that some adjustments were necessary to improve the game's functionality and player experience. For example, in our initial design (Phase 1), we planned to have individual classes for each type of tile entity, such as Ground, Reward, and Hole. However, during development, it became apparent that we could simplify this by handling all interactions within the Tile class itself. This change allowed us to control each tile's properties more efficiently and reduced unnecessary complexity in the code.

In developing this game, we chose to use the Abstract Factory design pattern to make the game more adaptable and manageable. This design pattern allowed us to create factories that could generate various elements like enemies, traps, rewards, and the player's character based on the

difficulty level chosen at the start. For instance, selecting Hard mode means the factory generates more enemies and traps while reducing the number of presents. With Easy mode, the factory produces fewer enemies and more rewards. This setup made it straightforward to tailor the game environment to each difficulty setting, keeping the code simple and ready for future expansions, such as adding new levels or more complex challenges. Collecting rewards, avoiding enemies, winning or losing, and encountering traps are all managed through objects created by the factory. For example, enemies in Hard mode can chase the player more aggressively, while bonuses like the "Disappearing Rudolph" appear for only a short time, encouraging quick collection. This approach enhanced gameplay depth and provided a solid foundation for future updates.

Management Process and Division of Roles

Effective management and teamwork were crucial to our project's success, especially since we relied mostly on text-based communication. We set up a group chat where each team member could easily discuss their progress, ask questions, and share ideas. To stay organized, we updated the group regularly on individual tasks and gave each other feedback. This setup allowed us to keep track of everyone's work and make any adjustments necessary to ensure the project stayed on schedule. We used this chat to go over any challenges or roadblocks and find solutions together, which helped us overcome obstacles more effectively. By focusing on clear and consistent communication, we were able to build a supportive environment where everyone felt comfortable asking for help or clarifying any details they needed.

Each team member had a specific role that suited their strengths and interests. They updated the documentation regularly with changes to class structures, methods, and any important adjustments we made to our initial design plan. This organization was key in helping us remember and understand our work, especially as the project grew more complex.

External Libraries Used

For our game, we relied primarily on Java's Swing for our graphical user interface (GUI). Swing is a powerful library built on top of the Abstract Window Toolkit (AWT) that provides a more flexible and sophisticated toolkit for creating GUI components. While we considered using AWT for certain elements, our decision to focus on Swing was based on its extensive features and capabilities that are better suited for our game's needs.

Firstly, as part of the standard Java Development Kit (JDK), Swing is readily available, which simplifies the setup process for our team. This accessibility allows us to focus on development without worrying about compatibility issues that may arise from using third-party libraries. Secondly, Swing is well-documented, with numerous online resources and community support

available. This extensive documentation has been invaluable, especially when we faced challenges during development, as we could quickly find examples and solutions to guide us.

Moreover, Swing offers a robust set of components for building interactive interfaces. It enables us to create windows, buttons, panels, and other GUI elements that are essential for our game. With Swing's lightweight components, we were able to achieve a polished look and feel, significantly enhancing the overall user experience. The customization options available in Swing allowed us to tailor the interface to fit the theme of our game, making it visually appealing and engaging for players.

Although we have primarily used Swing so far, we remain open to exploring other options in the future, such as JavaFX and AWT for specific functionalities or considering newer libraries that may offer additional features. This flexibility will allow us to adapt our approach as the project evolves and to ensure that we are using the best tools available for our development needs.

Measures Taken to Enhance Code Quality

To enhance the quality of our code during the development process, we implemented several best practices. First, we ensured that our code was well-structured and modular. This means breaking down complex functionalities into smaller, manageable classes and methods, making it easier to read, maintain, and debug. We also followed the principles of object-oriented programming (OOP), promoting reusability and abstraction. We established consistent naming conventions early on. For example, all classes used clear, descriptive names that represented their purpose, such as Tile for the tile class and Entity for objects within the game world. This consistency made the code more intuitive and helped avoid confusion, especially as the project grew in complexity.

To further enhance readability, we also used in-line comments to explain complex sections of code, especially in areas where multiple conditions needed to be checked, like in the occupy method of the Tile class. Additionally, we created Javadoc to document each class and method, which served as a guide for understanding the structure and functionality of our code. This documentation was especially useful when revisiting parts of the code for debugging or making improvements. Finally, we incorporated testing and debugging into our workflow to catch bugs early and verify that our code functioned as expected. By using JUnit tests, we could verify that each game function operated correctly, especially as we added new features.

Challenges Faced During This Phase

Phase 2 presented various challenges that tested our teamwork and technical skills. Time management emerged as one of the most significant hurdles. Each team member had their own

academic commitments and extracurricular activities that required careful balancing. This often led to conflicting schedules, making it difficult to coordinate meetings and progress updates. The pressure of deadlines intensified as we tried to align our efforts, resulting in a constant race against time to meet our objectives.

Another major challenge was determining the best way to design our classes and establish relationships between them. Early in the phase, we quickly recognized that having a clear and organized structure would be essential to keeping the game code manageable and scalable. Our initial draft for the game's design, based on the first-class diagrams, required significant revisions as we delved deeper into the project. We grappled with the intricate details of how each class would interact, often getting bogged down in discussions about the roles and responsibilities of various game components. This led to moments of confusion and frustration as we navigated the complexities of our design.

Determining how entities like players, tiles, and obstacles would connect and communicate within the game world proved to be a particularly challenging aspect. We frequently had to revisit our design documents to ensure that we were on the right track, which resulted in a lot of back-and-forth discussions that sometimes felt unproductive. The iterative nature of this process required us to be patient with each other and willing to adapt our ideas based on group consensus.

Additionally, we faced difficulties in selecting and implementing the right design pattern for creating game elements. With a plethora of different objects in our game—such as various tiles, obstacles, and rewards—we needed a flexible and efficient way to handle their creation. The pressure to make the right choice weighed heavily on us, as we understood that a poor decision at this stage could lead to more complications down the line. After considering several options and debating their pros and cons, we ultimately chose the Factory Method design pattern. While this choice brought some clarity, it also introduced its own complexities, as we had to ensure that all team members were on board with the implementation details.

Designing an intuitive user interface added another layer of complexity to our development process. We aimed to create a game that was not only enjoyable but also easy to navigate without excessive explanations. Striking the right balance between aesthetics and functionality proved to be more challenging than we anticipated. We spent considerable time selecting suitable images for tiles and characters, striving to ensure that each part of the game screen looked well-organized and visually appealing. This process involved multiple revisions and feedback sessions, as we wanted to ensure that our design met both our artistic vision and usability standards.