BLOB2D ARCADE

PROJECT REPORT

1. Implementation Approach

- Adapted unified process modal to carry-out the project.
- Design planned with regards to objectives and requirements.
- Schedule created for daily meetings and deadlines.
- Progress and problems discussed in the meetings.
- Discord and WhatsApp used for communication purposes.
- Divided tasks between the team members.
- Object oriented design implemented to structure the source code.
- Object classes implemented in individual files.
- Class attributes encapsulated to reduce error possibilities and unwanted access.
- Inheritance and polymorphism techniques used to increase code reusability and code repetition.
- Project automated to improve productivity.
- All the objects in the game (For example: Player, enemy, walls, etc.) considered as individual entities.

2. Adjustments and Modifications

Many adjustments were made in design and class diagram while implementing the game. Due to our limited knowledge in Java before the project started, we made lots of changes during the implementation. Better insight of requirements and objectives was gained during implementation, which resulted in adjustments in the design. New methods and attributes were added to the classes according to the requirements. Some methods were also lifted upper in the hierarchy to reduce code duplication. We were thinking to use Slick2D package for GUI, but we found JFrame and JPanel easier and better to use, so we decided using those instead of Slick2D. We also changed the game screen by getting a different background and different images for character images from what was decided in the design phase. We decided to add an exit screen which gives the player option to either play again, or to go back to the main menu.

User interface was edited to add game legends on the home screen. This was done to get the new users know about the characters of the game. Constant variables were introduced to increase code readability and to reduce errors. New objects ExitTile() and Gate() were added. Gate() allows the player to collect all the rewards first before ending the game. ExitTile() was implemented as a special tile to place the Gate abject.

3. Project Management

The project was managed using Gitlab for sharing progress. Daily meeting and deadlines were created in the schedule to complete the project timely. Problems and progress were discussed in the meetings everyday. Unified process model was implemented to carry-out the procedures. Every stage of the project was achieved timely in order to deliver the project before deadline. The roles were divided between the teammates as follows:

Max	**Michael**	**Ketan**
JFrame	Game board	Punishment
Game Timer	Player object	Bonus reward
GUI	Walls and barriers	Cake
User Inputs	Moving enemy	Reporting
Implementing timer	Switching Screens	Reward and punishment Placing

4. External Libraries

The following libraries were used for the project:

1. **Java.util:** To use ArrayList for storing objects and their attributes, and to use Randomization for random bonus reward positions. It was also used to getkeyboard input.

2. **Java.io:** To load image files of entities and for exception handling in case of errors. To handle system input and output through data stream.

3. **Java.awt:** To implement user interface and for painting graphics and images.

4. **Jawa.swing:** To implement timer for Bonus respawns, actions performed, game tick and GUI. For drawing images of entities on the screen.

5. **Jawa.imageio:** To locate ImageReaders and ImageWriters.

5. Quality Enhancement

Various techniques were used to enhance the quality of the code. Intensive analyzation was carried throughout the whole project. A specific process model was followed to deliver the project. Every state of the project model was utilized for progress. Regular communication was built throughout the states to ensure quality and timely progress. Project requirements were understood to be successfully achieved. Process automation was done increase productivity and to handle external library imports. Version control system was used to track and share changes and progress in the project. Inheritance and polymorphism used to reduce code redundancy and to increase reusability. Sufficient comments added to make sure that the code Is readable. Sensible names used for attributed, methods and classes to increase code understanding. Code was reviewed by other teammates to reduce errors and bugs. Design principles were considered while designing the GUI. Prototypes were created while the elaboration phase of the project to better understand the requirements, and then were elaborated and incremented over the construction phase of the project.

6. Challenges Faced

There were some challenges we faced during the implementation, but we overcame them by working together and helping each other. The initial challenge we faced was learning java. None of us was familiar or had java experience before this project. We spent couple weeks learning java by watching videos and reading resources online. Due to this, we faced a major delay in our project.

The other challenge we faced was implementing the moving enemy. We found it harder to make the enemy move on a shortest path towards the player after ever tick. It was colliding with walls, and sometimes passing through the wall to reach the player. We solved the problem by implementing a function

which finds the player's direction with respect to the enemy and sends the enemy one step forward in that direction while detecting the walls and other enemies at the same time.

The third and the last challenge faced was implementing switch screen feature in the game, which was needed to change the screen when starting and ending the game. The game timer was not properly syncing with the screen when it was switched.

Overall, we went through some hard problems while completing the project, but we eventually overcame all the problems working on them together. We learned a lot of new things while implementing the game and this project also enhanced our management skills.