# Software Requirement Specification
## Game Dev - Block Basher

**Team Name - Nebula Nexus**
Member 1 - M Srinivasan(IMT2021058)
Member 2 - P. Ramsai Koushik (IMT2021072)
Member 3 - Adithya Nagaraja - (IMT2021054)
Member 4 - Devendra Rishi Nelapati - (IMT2021076)

**Architecture -** We are planning on using layered architecture for this application. The details are as follows:

**Presentation Layer**: It interacts with the Application Layer through the GameEngine class to display game elements, scores, customization options, and difficulty levels.

**Application Layer**: This layer interacts with the Domain Layer through the BrickManager class to manage bricks and with the Infrastructure layer through InputConnector and CollisionConnector to handle user input and collisions. It also interacts with itself to control game elements and logic.

**Domain Layer**: It primarily interacts with the Application Layer to provide game entities like the Brick class.

**Infrastructure Layer**: It interacts with the Application Layer through InputConnector and the CollisionConnector to provide raw input and collision information. It also interacts with the Presentation Layer through the UserInterface class to handle error messages.

This layered architecture promotes modularity and separation of concerns, allowing for greater maintainability and extensibility of the system.

**Components: -** Components are self contained units of code that encapsulates a functionality of the game. They are designed to be reusable and help the programmers organize and manage the software codebase.

**1. Game Engine:**
- Responsible for handling game mechanics, physics, and overall game state.
- Connects to the Input Handler to receive player control inputs.

**2. Paddle Controller:**
- Manages the movement of the paddle based on user inputs.
- Connects to the Input Handler to receive control signals.

**3. Ball Controller:**
- Handles the movement and collision detection of the game ball.
- Interacts with the Game Engine to update the game state based on ball movements.

**4. Brick Manager:**
- Manages the arrangement, colors, and durability of bricks.
- Connects to the Collision controller to detect collisions and update brick durability.

**5. Scoring System:**
- Keep track of the player's score.
- Connects to the User Interface for real-time display of the current score.
- Connects to the Game engine for updating scores.

**6. Customization Manager:**
- Manages customization options for paddle design, ball color, and background theme.

- Connects to the User Interface to reflect user choices visually.

**7. Difficulty Manager:**
- Adjusts game difficulty based on the player's performance.
- Connects to the Game Engine to modify game parameters like ball speed and brick patterns.

**8. User Interface (UI):**
- Displays game elements, score, and customization options.
- Connects to the Scoring System for real-time score updates.
- Connects to the Customization Manager for reflecting user choices.
- Connects to the Difficulty Manager for displaying the current difficulty level.

**9. Input Handler:**
- Handles user inputs, such as keyboard inputs.

**10. Error Handler:**
- Manages errors and provides clear error messages to users.
- Ensures the game remains stable and reliable, handling unexpected issues gracefully.

## Connectors:

**1. Input Connector:**
- Connects the Input Handler to the Paddle Controller for transmitting user control signals.

**2. Collision Connector:**
- Connects the Ball Controller to the Brick Manager for detecting collisions and updating brick durability.

**3. Score Connector:**

- Connects the Scoring System to the User Interface for real-time display of the player's score.

## Relationship of Components and Connectors with FRs and NFRs

Functional Requirements (FRs) and Non-Functional Requirements (NFRs) are categories used to define different types of specifications for a system. Let's map the provided features (FRs) and constraints (NFRs) to the components and connectors discussed earlier in the context of the Block Basher game:

**Functional Requirements (FRs):**

**1. Game Mechanics:**

- **Components**:
  - Game Engine: Responsible for implementing classic brick breaker mechanics, ball physics, and paddle movement.
  - Paddle Controller: Manages player input for controlling the paddle.
  - Ball Controller: Handles ball movement and collision detection.
  - Brick Manager: Manages brick arrangement, colors, and durability.
- **Connectors:**
  - Input Connector: Connects the Input Handler to the Paddle Controller for transmitting user control signals.
  - Collision Connector: Connects the Ball Controller to the Brick Manager for detecting collisions and updating brick durability.

**2. Scoring:**

- **Components:**
  - Scoring System: Keeps track of the player's score.
  - User Interface: Displays the player's score.
- **Connectors:**
  - Score Connector: Connects the Scoring System to the User Interface for real-time display of the player's score.

3. **Customization Options**:
   - **Components**:
     - Customization Manager: Manages paddle design, ball color, and background theme options.
     - User Interface: Displays customization options.
   - **Connectors**:
     - Customization Connector: Connects the Customization Manager to the User Interface for reflecting user choices visually.

4. **Adaptive Difficulty:**
   - **Components:**
     - Difficulty Manager: Adjusts game difficulty based on the player's performance.
     - Game Engine: Incorporates changes in game parameters.
   - **Connectors:**
     - Difficulty Connector: Connects the Difficulty Manager to the Game Engine for adjusting game difficulty parameters.

**Non-Functional Requirements (NFRs):**

1. **Performance**:
   - **Components:**
     - Game Engine: Ensures smooth gameplay and accurate physics calculations.
   - **Connectors:**
     - Input Connector: Should transmit user inputs efficiently.
     - Collision Connector: Should handle collisions accurately and efficiently.

2. **User Interface:**
   - **Components:**
     - User Interface: Should be intuitive, visually appealing, and easy to navigate.
     - Error Handler: Should provide clear instructions and feedback, handling errors gracefully.

**- Connectors:**
- Score Connector: Should update the User Interface with minimal latency.

**3. Reliability and Stability:**

**- Components:**
- Error Handler: Ensures stability and handles errors gracefully.

**- Connectors:**
- Error Connector: Connects the Error Handler to various components for handling errors.

Each FR is mapped to specific components responsible for fulfilling that requirement, and connectors facilitate communication between these components. Similarly, NFRs are considered during the design and implementation of both components and connectors to ensure the overall performance and reliability of the Block Basher game

**Detailed Design:**
In a Brick Breaker game with multiple kinds of bricks and varying block difficulty, you'll need data structures to represent the game state, including the arrangement of bricks, their types, and their difficulty levels.

**1)Data Structure:**
- **HashMap :** This HashMap stores the value of color codes of the bricks.
- **Array[][]:** To represent the game grid where each cell corresponds to a brick. The value of the array stores the hardness (color) of brick

**2) Algorithms:**
- **Collision Detection Algorithm:**It checks for collisions between the ball and the paddle, as well as between the ball and the bricks. If a

collision is detected with a brick, the brick is marked as destroyed, and the ball's direction is adjusted based on the collision point.

- **Game Over Algorithm**: If the ball reaches the bottom of the screen, the game is over, and a game over message is displayed. If all bricks are destroyed, the game is over, and a game over message is displayed.
- **Map Initialization Algorithm:**This generates the initial map of the level, it stores the width, height and color of each brick based on the level and difficulty.
- **Ball Movement Algorithm:** This updates the position of the ball based on its current direction. It also changes the values accordingly after collisions with the walls and adjusts the direction accordingly.

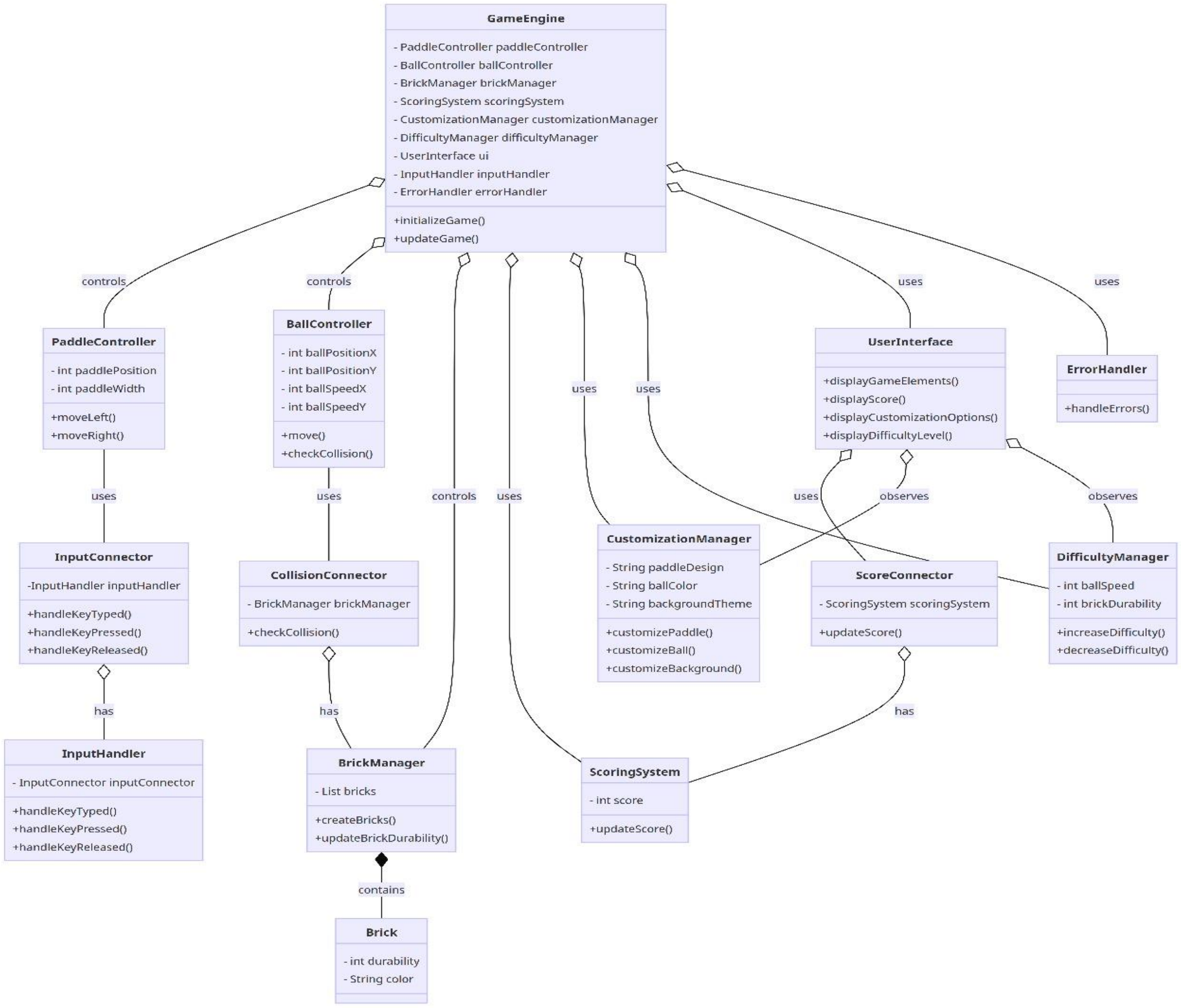
## 3)API's:

- **Swing API:**
  **javax.swing.JFrame:** The JFrame class is part of the Swing framework and is used to create the main window for the game.
- **AWT API (Abstract Window Toolkit)**:
  **java.awt.Graphics and java.awt.Graphics2D**: These classes are used for drawing shapes and rendering graphics on the panel.
  **java.awt.event.KeyListener**: Used for handling keyboard events.
  **java.awt.Color**: Used for defining colors when drawing graphics elements on the screen.

| **Component** | **Functionality** | **Connectors** |
|---|---|---|
| GameEngine | Initializes the game, updates the game state, and renders the game to the user interface | |
| PaddleController | Controls the movement of the paddle | InputHandler |
| BallController | Controls the movement of the ball | |
| BrickManager | Manages the bricks in the game | |

| | | |
|---|---|---|
| ScoringSystem | Keeps track of the player's score | GameEngine |
| CustomizationManager | Allows the player to customize the game | InputHandler |
| DifficultyManager | Manages the difficulty of the game | InputHandler |
| UserInterface | Displays the game to the user and handles user input | GameEngine, InputHandler |
| InputHandler | Handles keyboard input from the user | PaddleController, CustomizationManager, DifficultyManager |
| ErrorHandler | Handles any errors that occur during the game | GameEngine |

## Class diagram -



**Figma Design:**
https://www.figma.com/file/FJ1goFz2bxjMVnrSNkafqX/Block-Basher?type=design&node-id=0%3A1&mode=design&t=4nT7FZCYOnWOF83k-1

**Work Split:**

**Architecture:** Srinivasan
**Detailed Design:** Adithya
**UI Design:** Ramsai Koushik and Rishi
**Class diagram:** Srinivasan, Adithya and Ramsai Koushik