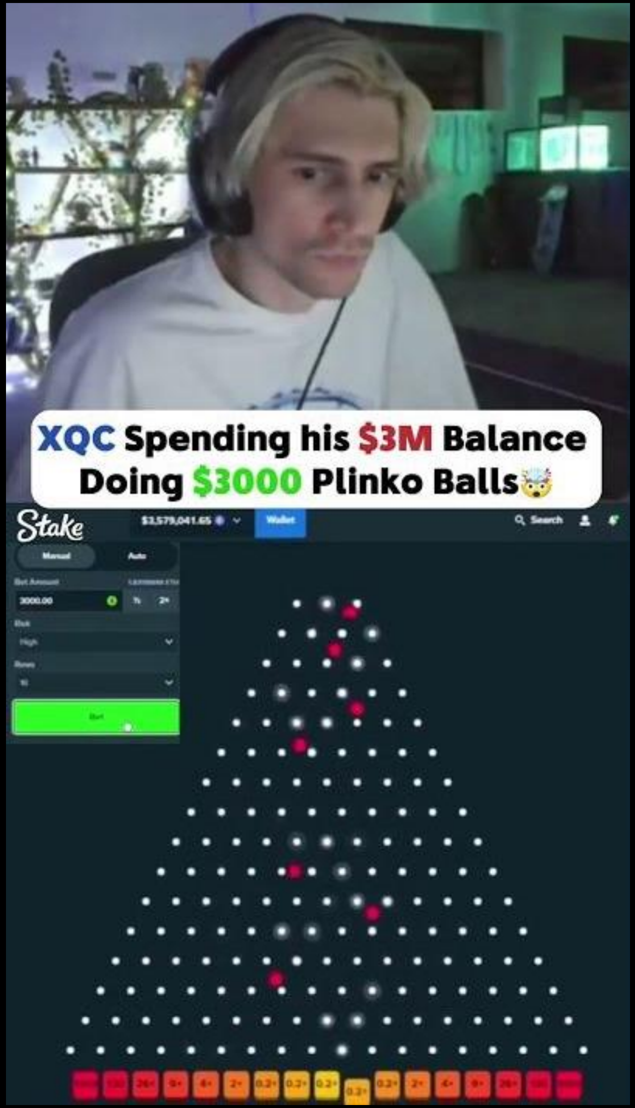
Plinko

Team Name: Good Noodles

Members: Mark Sanghera, Jack Kabil, Scott Riddle

Game description

- Plinko is a gravity-based game where a player drops a ball into a system of pegs
- User bets a certain amount on each ball
- The dropped ball then bounces off the pegs until it reaches the bottom
- Buckets at the bottom contain multipliers
- Outer multipliers are > 1, inner are < 1
- User is returned betted amount multiplied by multiplier it lands on
- Single player



Project Overview

- Features:
  - **Physics Engine:** We used the dyn4j 2d engine to implement the physics
  - **Assigning monetary value to each ball:** The dropped ball requires a monetary value so that it can be multiplied by the zone it lands in
  - **Player balance amount:** The bets placed by the player will come from their balance and any money that is lost or won will be added to the balance
  - **Slot Rewards:** When a dropped ball lands in a slot, a multiplier will be applied to the monetary value of the dropped ball

- Limitations:
  - **Physics Engine:** Difficulty in learning how it works and how to connect to Java Swing
  - **Java Swing:** Very clunky and hard to work with

Rules

- Cannot send a ball greater than your balance

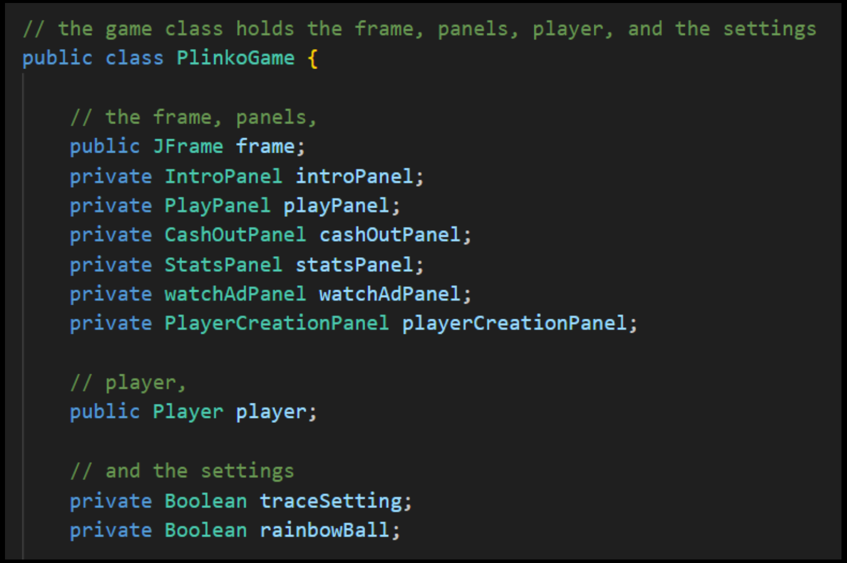
Project Requirements

- Functional Requirements
  - Working physics
  - Betting / gambling
  - Player bank account affected

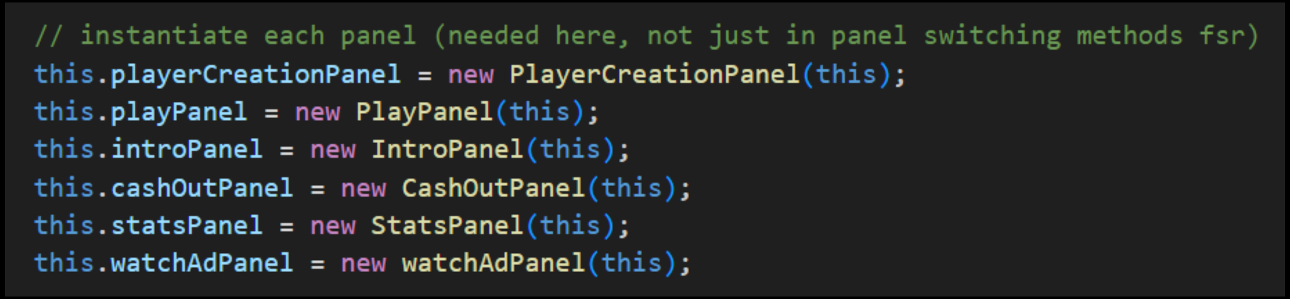
- Non-Functional Requirements
  - Clean UI
  - Some customization / settings
  - Fun additional features

Project Solution Approach

- Each panel extends JPanel and is added to a JFrame
- Class PlinkoGame contains the frame, panels, settings
- PlinkoGame passed into each panel in order to access settings
- Implemented image IO for images
- Also used java.io to store players' names and bank accounts with CSV

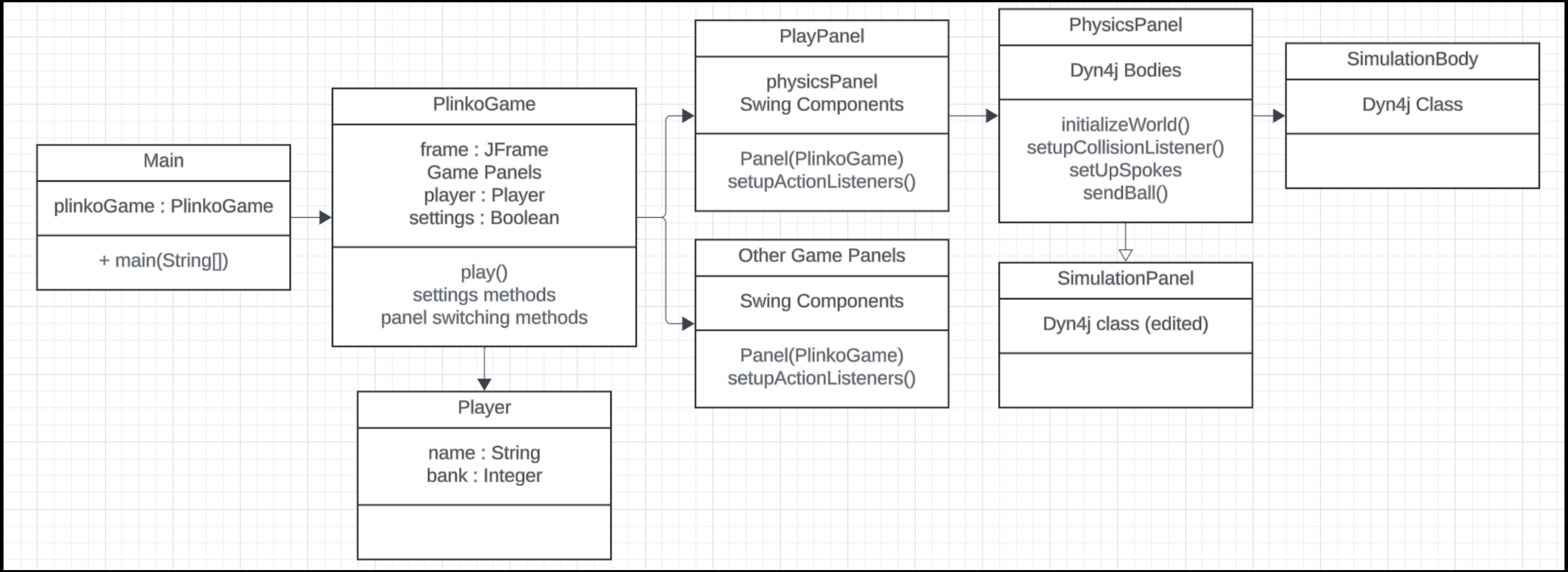


```java
// the game class holds the frame, panels, player, and the settings
public class PlinkoGame {

    // the frame, panels,
    public JFrame frame;
    private IntroPanel introPanel;
    private PlayPanel playPanel;
    private CashOutPanel cashOutPanel;
    private StatsPanel statsPanel;
    private watchAdPanel watchAdPanel;
    private PlayerCreationPanel playerCreationPanel;

    // player,
    public Player player;

    // and the settings
    private Boolean traceSetting;
    private Boolean rainbowBall;
```



```java
// instantiate each panel (needed here, not just in panel switching methods fsr)
this.playerCreationPanel = new PlayerCreationPanel(this);
this.playPanel = new PlayPanel(this);
this.introPanel = new IntroPanel(this);
this.cashOutPanel = new CashOutPanel(this);
this.statsPanel = new StatsPanel(this);
this.watchAdPanel = new watchAdPanel(this);
```

UML Class Diagram



Team Collaboration Approaches

- Primary communication was done through discord, secondary was mostly in person
- We did not have any issues with GitHub
- We did lots of branching at the start, but towards the end we stopped branching
- Biggest lesson we learned as a group was communication was key
- We did a few group hacking sessions but it is was mostly individual*.

Testing, Validation, and Acceptance Plan

- What testing approaches do you plan on using?
  - We primarily used Integration, Functional, and User tests
    - Primarily, is the physics engine connected to the rest of the game?
  - We had 1 unit test to test the Stats panel

- Our project is deliverable because the game works as intended and it never crashes

- "It just works" -- Todd Howard

Click to add title

WHO IS READY TO GAMBLE

Summary

- Java swing is difficult to use and took more time than the logic
- Dyn4j (physics engine) wasn't any better
    - Hard to debug
    - Cryptic documentation
    - Sparse / weak comments in source code

dyn4j