Name : Vedika Jain and Zeel Gajjar Friday, 17th October, 2025

## 1. Project Summary

Our project idea : Zombied is a space-themed educational game where players learn physics by launching projectiles at target planets.

Purpose of our project : To teach and reinforce physics concepts, particularly projectile motion through game-play

Theme : Space and planets, creating a fun cosmic environment

Main features : Adjustable projectile launches using real physics scoring and levels to track progress with time restraints; feedback on miss hits to reinforce learning.

Features and its MVC relations:

- **Rocket Launch Simulation**:
    - **Model**: *GameModel.simulateLaunch()* uses *Projectile*, *PhysicsUtil.calculateTrajectoryPoint()*, and *CollisionUtil.checkCollision()* to determine if the rocket escapes currentPlanet and reaches targetPlanet; Trajectory tracks path and failureReason for "Zombied".
    - **View**: GameView animates Trajectory.path on solarSystemCanvas and shows input controls (speedInput, angleSlider).
    - **Controller**: *GameController.handleLaunch()* processes inputs and *updateGameState()* sets isZombied.
- **Level Progression**:
    - **Model**: *GameModel.advanceLevel()* updates currentLevel, targetPlanet, and difficultyFactor.
    - **View**: GameView updates levelLabel.
    - **Controller**: *GameController.handleNextLevel()* triggers level advance.
- **Time Limit Mechanic**:
    - **Model**: GameModel.levelTimeLimit and *getRemainingLevelTime()* track countdown; isZombied set on timeout.
    - **View**: GameView displays timeLabel.
    - **Controller**: *GameController.startCountdown()* and *checkCountdown()* manage Timeline.

- **Scoring and Game State**:
    - **Model**: GameState tracks score, attempts, and totalPlayTime via *GameModel.updateGameState()*.
    - **View**: GameView shows scoreLabel; ResultView displays final score.
    - **Controller**: *GameController.showResult()* updates based on GameModel.
- **User Interface**:
    - **Model**: Provides data (e.g., Planet positions) to views.
    - **View**: GameView renders planets and controls; MenuView shows buttons; ResultView displays outcomes.
    - **Controller**: *MenuController.handleStart()* switches scenes; *ResultController.handleReturnToMenu()* navigates.
- **Audio Feedback**:
    - **Model**: N/A (data-driven by GameController).
    - **View**: N/A (audio is non-visual).
    - **Controller**: *SoundController.playLaunchSound()*, *playHitSound()*, *playMissSound(), mute(), adjustVolume(), toggleMute()* triggered by GameController.
- **Result Feedback**:
    - **Model**: GameModel.isZombied and Trajectory.failureReason provide outcome data.
    - **View**: *GameView.showZombied()* alerts; *ResultView.displayOutcome()* and *displayExplanation()* show details.
    - **Controller**: *GameController.showResult()* and ResultController manage display.

**2. Task Breakdown (Work Breakdown Structure)**

- Models to implement
    - Game model → core logic of the game; handles projectile simulation, level progression and scoring
    - GameState model → tracks and update the player’s progression & attempts
    - Projectile model → stores physics values (speed, angle position) and calculates trajectory
    - Planet model → represents target planets with position and hit detection

    - Trajectory model → records projectile's path and determines the reason of the failure
- Views to implement
    - Menu View (menu.fxml) → main menu of the screen, use CSS for colors, fonts, and hover effects
    - Game View (game.fxml) → Main gameplay interface; animates projectile trajectory and displays score, level, timer, and controls (solar system canvas), CSS used for background color, labels and button styling
    - Result View (result.fxml) → displays final score and explanation. CSS styles messages and buttons consistently in the theme.
- Controllers to implement
    - MenuController → handles the main menu
    - GameController → manages game play logic, processes user input (speed/angle), launches projectiles, updates GameState
    - ResultController → Displays results and explanations at the end of the game, and handles navigation back to the menu
    - SoundController → Plays sound effects for launches, hits and misses and manages audio setting like mute and volume adjustment
- Utilities to implement
    - PhysicsUtil → Contains methods for projectile motion and trajectory calculations.
    - CollisionUtil → Determines whether a projectile collides with a planet.
- Testing tasks
    - Writing and testing JUnit test for the models and utilities.
    - UI testing → Verify that all interface components (buttons, sliders, and inputs) trigger the correct events .

- Deployment Tasks : Package the project into an executable (.jar or similar), ensuring all assets (FXML files, CSS, images, and sounds) are included. Test the packaged version to verify that it runs properly outside the development environment.

- Documentation
    - Create class and method documentation → Add Javadoc comments explaining each class, method, and attribute.

    - Testing summary → Include a section describing how the project was tested and key results.

## 3. Responsibilities and Task Allocation

Documentation must be completed within each classes' mid-week deadline.

Testing and deployment tasks must be completed with its own applicable classes.

| Module/Task | Team Member | Deadline |
|---|---|---|
| Zombied | Vedika | October 24;<br>Mid-deadline: October 22 |
| GameController | Zeel | October 24<br>Mid-deadline: October 22 |
| GameView and GameObserver | Vedika | October 31<br>Mid-deadline: October 29 |
| GameModel | Zeel | October 31<br>Mid-deadline: October 29 |
| GameState and Planet | Vedika | November 7<br>Mid-deadline: November 5 |
| Projectile and Trajectory | Zeel | November 7<br>Mid-deadline: November 5 |
| SoundController | Vedika | November 14<br>Mid-deadline: November 12 |
| MenuController and MenuView | Zeel | November 14<br>Mid-deadline: November 12 |
| ResultView | Vedika | November 21<br>Mid-deadline: November 19 |
| ResultController | Zeel | November 21<br>Mid-deadline: November 19 |
| Add CSS for styling the MenuView | Zeel | November 14 |
| Add CSS for styling the GameView | Vedika | October 31 |
| Add CSS for styling the ResultView | Zeel | November 21 |

## 4. Trello Board Integration

Trello: https://trello.com/b/d8qZsXB4/sem-3-programmingfinalproject
Github: https://github.com/ZeelDGajjar/Sem3-Final_Project.git

## 5. Risk Management

Problem 1: Scheduling Delays
Some classes or components may take longer than a week to implement, which can push back integration and testing. This could delay overall progress and affect Friday deadlines.
Solution:
Set mid-week checkpoints to track progress and adjust plans early. Break larger tasks into smaller steps and prioritize critical components first. Prepare questions in advance to resolve blockers quickly during check-ins or with external help if needed.

Problem 2: Technical Integration Issues
Different MVC components may not work together smoothly at first, leading to bugs or compatibility issues during integration.
Solution:
Schedule dedicated integration and testing time each week. Use Git for version control to manage changes and prevent conflicts. Write and run unit tests early to identify issues before full integration.

Problem 3: Teamwork and Communication Gaps
Working on separate components can lead to miscommunication or uneven workload, especially if one part takes longer than expected.
Solution:
Hold short, regular check-ins to share progress and challenges. Document responsibilities clearly and stay flexible to redistribute work when needed. Prepare and list any technical or design questions ahead of meetings to keep communication focused and efficient.