2024

StartDocument

Prepared by:
Abu Hasib Shanewaz
Ayodeji Olagoke
Kaiser Aftab
Precious Oziwo

C# 2 PROJECT

NHL STENDEN
university of applied sciences

Contents

Version Control

Version No.	Changes
1.0	Created document and filled background information chapter
1.1	Filled in rest of document
1.2	Edited chapters

Background Information

Game Overview

The game is designed around a central, player character who must defend against increasingly difficult waves of enemies until reaching a climactic boss fight. The gameplay is anchored in a strategic combination of real-time defense and skill management, encouraging players to think tactically about their skill upgrades and use of in-game resources.

Core Mechanics

- **Enemies**: Various enemies with distinct shapes and abilities will spawn around the player. Each type of enemy will have different patterns.
- **Skill Trees**: Two distinct skill trees provide a pathway for players to enhance their abilities. Players can earn upgrades through gameplay achievements and combine different skills to tailor their defensive capabilities to their preferred play style.

Objective

The primary objective of the game is to survive through the waves of enemies and ultimately defeat the final level. Success in the game is measured by the player's ability to effectively manage their skills and resources, making strategic decisions to enhance their chances of survival.

Features

- **Endgame Boss Fight/ level**: The culmination of the game is a boss fight, presenting a significant challenge that requires players to utilize all the skills and strategies they have developed throughout the game.
- **Skill Combination and Customization**: The dual skill trees allow for extensive customization and strategic diversity, letting players experiment with different combinations of abilities to best suit their play style.

User Interface and Interaction

- **Start and End UI**: The game will feature a comprehensive start menu and an end-game screen, which includes options to restart the game, access the leaderboard, and modify game settings.
- **Leaderboard**: A leaderboard system will track, and display player scores based on survival time and efficiency, fostering a competitive environment.

- **Music and Sound Controls**: Players will have the ability to control music and sound effects from within the game, enhancing the immersive experience of the gameplay.

Mock-ups and detailed information

The player seeks to protect his house from waves of enemies.

The player can pause and interact with a menu

After a skill tree has been maxed out, the player will face a final challenge as the game's finale.

Feasibility:

Feasibility

We assessed the feasibility of developing our proposed game. Here is a concise and clear explanation of our findings.

Technical Feasibility

1. Tools and Platform:

- We will employ C# and the .NET framework to construct the game, as it corresponds to the specifications of our assignment.
- Our development environment will be Visual Studio 2019 or a newer version, which will provide us with the necessary tools for coding and debugging.
- Instead of utilizing Unity, which is prohibited, we will explore the possibility of employing MonoGame and Wpf using the MVC architecture.

2. Team Skills:

- Our team possesses substantial expertise in C# programming, gained from prior courses and projects.
- We will utilize internet lessons and resources to acquire the necessary proficiencies in certain frameworks.

3. Game Features:

- The game will incorporate skill progression systems.
- To ensure code organization and scalability, we will utilize object-oriented programming to incorporate different enemy kinds

Operational Feasibility

1. Development Timeline:

- We possess a well-defined strategy encompassing research, prototyping, the creation of essential functionalities, and concluding with comprehensive testing.
- Regular assessments and feedback meetings will ensure that we adhere to the timetable and make any required modifications.

2. Team Collaboration:

- We have established efficient communication channels and precisely delineated responsibilities for each team member.
- GitHub will be utilized for version control to effectively manage code modifications and facilitate seamless collaboration.

In conclusion

Our investigation indicates that the development of this game is achievable. We possess the appropriate tools, expertise, and a well-developed strategy to effectively accomplish the project. Through the utilization of accessible resources and the establishment of strong team communication, we possess a high level of assurance in our ability to produce an enjoyable and fully operational game that fulfills all the stipulated assignment criteria.

Test Plan:

This test plan aims to confirm that the game meets design requirements, functions as intended and is user-friendly in every way. This entails evaluating the user interface, gaming mechanics, performance of the game, and the integration of various game elements.

Test Environment

- **Software**: Windows 11, MonoGame Framework, Visual Studio 2022
- **Hardware**: Standard PC with a graphics card capable of supporting MonoGame
- **Tools**: Unit testing frameworks (NUnit), MonoGame Testing tools

Test Cases

1. **Game Mechanics**

- **Player Mechanics**
    - Ensure the player can interact with the game's world correctly
- **Enemy Mechanics**
    - Test different enemy types for correct spawning behavior around the player.
    - Check that enemy attack patterns and interactions with the player are according to design specifications.
    - Validate the scaling difficulty of enemy waves as the game progresses.

- **Skill Trees**
    - Verify that skills can be upgraded and combined properly.
    - Test the effects of each skill upgrade to ensure they work as intended.

2. **UI/UX Testing**

- Menu Navigation
    - Test the functionality of all UI elements in the start and end menus.
    - Ensure that options to start the game, adjust settings, and exit are working correctly.
- Leaderboard
    - Validate that the leaderboard correctly displays user names, play times, and scores.

Sound and Music

- Test that background music and sound effects are playing correctly.
- Verify the functionality of music control options like play, pause, and volume adjustments.

3. **Performance Testing**
    - Monitor game performance to ensure it runs smoothly on the intended hardware.
    - Check for memory leaks or any performance issues during extended gameplay sessions.
4. **Integration Testing**
    - Ensure that all parts of the game, including gameplay mechanics, UI elements, and backend systems, work together without issues.

Moscow Analysis:

Must Have:

Player spawning.

Enemies spawning around the player with increasing difficulty.

Two distinct skill trees with upgradable and combinable skills.

Database to save and display usernames, playtime, and best times.

Start menu with basic options (play, settings, exit)

Additional enemy types with unique behaviours and utilities.

Should Have:

Mini-bosses or elite enemies to increase challenge.

Basic music control (play, pause or volume).

Could Have:

More skill trees and abilities to provide greater variety.

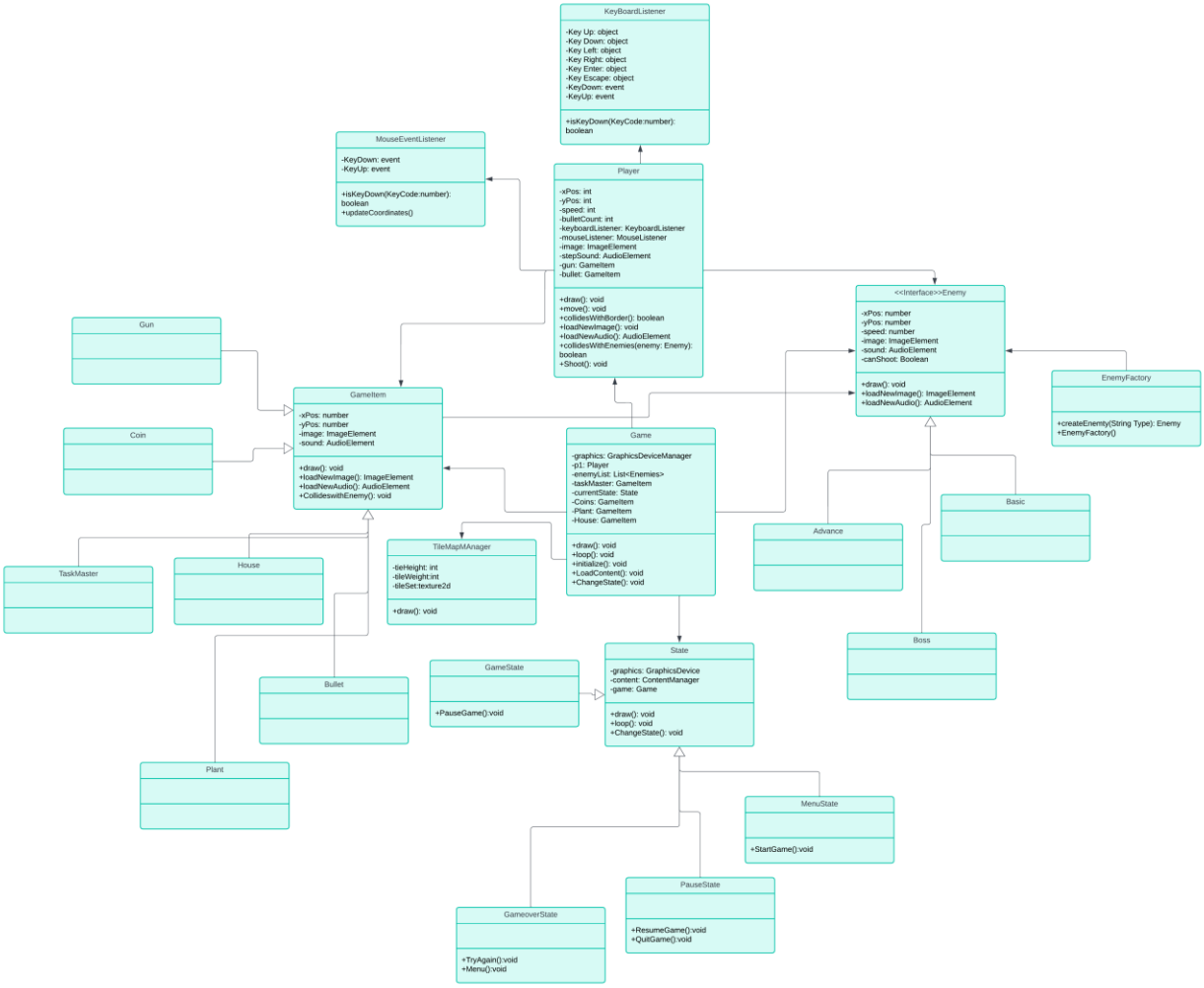
A currency system.

Won’t Have:

Online multiplayer

Boss fight to conclude the game.

UML diagram:


KeyBoardListener
-Key Up: object
-Key Down: object
-Key Left: object
-Key Right: object
-Key Enter: object
-Key Escape: object
-KeyDown: event
-KeyUp: event
+isKeyDown(KeyCode:number): boolean
MouseEventListener
-KeyDown: event
-KeyUp: event
+isKeyDown(KeyCode:number): boolean
+updateCoordinates()
Player
-xPos: int
-yPos: int
-speed: int
-bulletCount: int
-keyboardListener: KeyboardListener
-mouseListener: MouseListener
-image: ImageElement
-stepSound: AudioElement
-gun: GameItem
-bullet: GameItem
+draw(): void
+move(): void
+collidesWithBorder(): boolean
+loadNewImage(): void
+loadNewAudio(): AudioElement
+collidesWithEnemies(enemy: Enemy): boolean
+Shoot(): void
<<Interface>>Enemy
-xPos: number
-yPos: number
-speed: number
-image: ImageElement
-sound: AudioElement
-canShoot: Boolean
+draw(): void
+loadNewImage(): ImageElement
+loadNewAudio(): AudioElement
EnemyFactory
+createEnemy(String Type): Enemy
+EnemyFactory()
Gun
GameItem
-xPos: number
-yPos: number
-image: ImageElement
-sound: AudioElement
+draw(): void
+loadNewImage(): ImageElement
+loadNewAudio(): AudioElement
+CollideswithEnemy(): void
Coin
Game
-graphics: GraphicsDeviceManager
-p1: Player
-enemyList: List<Enemies>
-taskMaster: GameItem
-currentState: State
-Coins: GameItem
-Plant: GameItem
-House: GameItem
+draw(): void
+loop(): void
+initialize(): void
+LoadContent(): void
+ChangeState(): void
Basic
Advance
TaskMaster
House
TileMapMAnager
-tileHeight: int
-tileWeight:int
-tileSet:texture2d
+draw(): void
Boss
Bullet
GameState
+PauseGame():void
State
-graphics: GraphicsDevice
-content: ContentManager
-game: Game
+draw(): void
+loop(): void
+ChangeState(): void
Plant
MenuState
+StartGame():void
PauseState
+ResumeGame():void
+QuitGame():void
GameoverState
+TryAgain():void
+Menu():void

References:

1. Chacon, S., & Straub, B. (2014). Pro Git (2nd ed.). Apress. https://git-scm.com/book/en/v2
2. Godot Engine Documentation. (n.d.). Retrieved from https://docs.godotengine.org/en/stable/
3. Microsoft. (2019). Visual Studio 2019. Retrieved from https://visualstudio.microsoft.com/vs/
4. MonoGame Documentation. (n.d.). Retrieved from https://docs.monogame.net/