The main part

List of terms and definitions

Tower defense (TD) is a subgenre of strategy games where the goal is to defend a player's territories or possessions by obstructing the enemy attackers or by stopping enemies from reaching the exits, usually achieved by placing defensive structures on or along their path of attack. This typically means building a variety of different structures that serve to automatically block, impede, attack or destroy enemies.

A game engine is a software framework primarily designed for the development of video games and generally includes relevant libraries and support programs such as a level editor. The "engine" terminology is akin to the term "software engine" used more widely in the software industry.

Game engine can also refer to the development software supporting this framework, typically a suite of tools and features for developing games.

Description of business roles

Kozubenko Andrew - Full-stack
Polyachkov Dmitriy - Full-stack
Yakutina Svetlana - Full-stack
Primakova Julia - Full-stack

Functional requirements:

1. *Tower Editor*

- Setting the price of buildings and improvements.
- Determining the type of damage and its features (for example, magic damage, physical damage, etc.).
- Setting the shooting speed.
- The ability to create improvement chains/trees/graphs of tower improvements.
- Defining the visual elements of the towers, including animations and effects.

2. *Enemy Editor*

- Setting the number of life points, protection and speed.
- Identify strengths and weaknesses against certain types of damage.
- Customize the loot that enemies leave behind when destroyed.

3. *The technological tree*

- The developer can create and edit a global technology tree that will unlock new types of towers and upgrades throughout the game.

4. *Map Editor*

- Setting up enemy movement paths.
- Determining the points where the player can build towers.
- Setting goals (for example, the health of the base).
- Setting scenarios for the appearance of enemies.
- Options for controlling waves of enemies (the player can manually launch a wave).

- The ability to set winning or losing conditions (including endless modes).

5. *Launch and play*

The player can:

- Select a card and start the game.
- Build and improve towers on the map.
- Manage towers (set priorities, select targets, etc.).
- Observe the movement of enemies and interact with towers.
- Launch and control waves of enemies.
- Exit the game and save the results.

6. *Rewards and progress*

- The opportunity to receive rewards for successfully completing maps.
- Support for maintaining progress in the technology tree.

Non-functional requirements:

1. *Scalability*

- The system should support the creation of multiple towers, enemies, and maps without significant performance degradation.
- The engine should be designed in such a way that it is easy to support multiplayer modes in the future (the second stage of the project).

2. *Performance*

- The loading time of the map should not exceed 30 seconds.
- The response time to the player's actions (tower construction, improvement) should not exceed 0.1 seconds.
- The system must support simultaneous display of at least 50 active objects (enemies, towers, etc.).

3. *Ease of use*

- The interface of the map editor, towers and enemies should be intuitive for game developers.

4. *Extensibility*

- The engine should make it easy to add new types of towers, enemies, and maps without the need for significant code changes.
- The engine should support the possibility of further expansion of the functionality at the second stage of the project (adding a network game).

5. *Reliability*

- In case of a game or system crash, the user's data should be saved automatically.
- The engine must ensure that all data and repeats are saved in case of a critical error.

6. *Security*

- The game engine must be protected from making unauthorized changes to game data (for example, manipulating points or rewards).