Chicken Zombie Bonanza
Low Level Design
COP 4331C Processes of Object Oriented Software Fall 2011

Team 19 - Team (Cauc)asians

Team Members:
- Bernard Feeser
- Jon Leonard
- Danh Nguyen
- Jolene Wan
- Juan Chen

Modification history:

Version	Date	Who	Comment
v0.0	08/15/00	G. H. Walton	Template
v1.0	11/04/11	Jolene Wan	Trace of Requirements to Design
v2.0	11/04/11	Jon Leonard	For Deliverables 2

Contents of this Document

Design Issues
Detailed Design Information
Trace of Requirements to Design

Reference Documents
    Low Level Design Diagram Image

Design Issues

As discussed in the High Level Design document, one of our main issues with this design is concerned with how the player will play this game. Due to the mobile nature of this game, the player will be constantly moving around while checking the Android device periodically. This could lead to the player being distracted, which could result in bodily injury as well as other dangerous situations. To reduce the risk of this happening, we will be issuing a warning message at the start-up of the application. The message will warn the player to be more cautious while playing.

Another possible issue this design brings concerns the simplicity of the architectural model. Our model is fairly simple, with general classes that each encompass several different objects. For example, a GameEntity class refers to checkpoints, power-ups, enemies, etc. and a LifeformEntity class refers to the player as well as enemies. The purpose of this design approach is to increase the flexibility of the design. During the coding stages, decisions to change the structure of the design are imminent; making the architectural model more generalized makes making changes easier without having to travel back to earlier stages in our development model (the V-model).

Detailed Design Information

Reference the Low Level Design Diagram image.

Trace of Requirements to Design:

Requirements listed from SRS:

1. The application will take in GPS data.
   When the game starts, the GPSListener class will implement the PositionPublisher class to receive the GPS location from the Android device.

2. The application will display a map.
   The GameManager class, which will contain the main function, will take the API key from the Android device and publish a map on screen using the NavigationGame Google API.

3. The application will take in GPS data and display the user's location on map.
   The GameManager will create an GameEntity object to represent the player.
   The PositionPublisher will receive the player's position from the GPSListener and set the player's GameEntity position.
   The NavigationGame will get the player's GameEntity position and display the location on the map.

4. The application will define the game's play area.
   The GameManager will implement the game settings at the beginning of the game depending on the player's choice in the settings menu.
   The NavigationGame will pull the game area setting from the GameSettings and define the game's play area accordingly.

5. The application will run on Android Operating System 2.2.
   Hardware requirement; does not apply to high-level design.

6. The application will use the gyroscope to determine the device's orientation.
   The GyroscopeListener will be a OrientationPublisher that will allow OrientationListeners to listen for orientation updates, while it polls for changes in the device orientation.

7. The player will be able to use the device's orientation to view a complete 3D space.
   The ShootingGame will get the device's orientation from the OrientationPublisher and simulate a 3D space on the screen accordingly while a ShootingGame is the active game being played.

8. The player will be able to use the phone's orientation to generate weapon fire in the center of the device's screen.
   The OrientationListener will be implemented by the player's GameEntity.
   The ShootingGame receives this orientation to fire the active WeaponInventoryObject of the player.

9. The application will generate and display power ups on the map, within the play area.
   PowerUpEntity objects are created by the game manager which will then be displayed on the map of the NavigationGame.

10. The application will store the player's score for a game session when the program exits
    At the end of every game, the GameManager will store the active GameScore before exiting the program.

11. The player will be able to obtain power ups in the map view.
    The map of the NavigationGame will display PowerUpEntities.
    When a player moves close enough to a power up, the power up will active an ActivationListener and add to the inventory as an InventoryObject. The NavigationGame will then remove the power

up from the map and display it inside the player's inventory.

12. The application will generate and display a waypoint on the map at the start of the game, within the play area.
    WaypointEntity will be created by the GameManager and will then be displayed on the map of the NavigationGame.

13. The player will be able to activate a waypoint by going to its physical location.
    The map of the NavigationGame will display the WaypointEntity objects and when the player's GameEntity activates the WayPointEntity ActivationListener.

14. Zombie Chickens will charge towards the player in the FPS mini game.
    The game manager will create lifeform entities of the chickens.
    The PositionPublisher of the player will determine which direction the zombie chickens move during the shooting game when they charge at you.

15. Player health will change depending on how many zombie chickens hit you, and how many power ups you have collected.
    The player will be a LifeformEntity object, which implements the HasInventory class. Therefore it will have an inventory that includes InventoryObjects such as power ups. It will also be displayed on screen by the ShootingGame. The player, being a LifeformEntity, will also have a health bar. When a zombie chicken's position is close enough to the player, he/she will take damage and his/her health will decrease. If the player has any power ups that are HealthInventoryObjects, he/she will take less damage and the ShootingGame will remove the health object form the player's inventory.

16. The player will be able to play the shooting game.
    The game manager will implement the navigational game and return when a player reaches a waypoint to the game manager .
    The game manager will then generate an instance of a ShootingGame.

17. Players bullets will remove or weaken zombie chickens.
    Then the player reaches a checkpoint, the game will commence into a shooting mini game, where the ShootingGame class will take over. The player will then have infinite handgun bullets, as well as any other weapons they may have collected during the navigation game.
    When the player touches the screen, a bullet will be used from the active weapon and the health of any intersected chicken will be decreased by the weapon's specified damage level. If the health of the chicken falls equal or below zero, the (undead) chicken will die (again) and the GameManager will remove it from the screen and the map.

18. The player will be able to view their previous session statistics.
    The game manager will be used to view previous session statistics through an option to view past scores.

19. Activating a waypoint will start the shooting game.
    The game manager will start the navigation game which in turn will return when a waypoint is reached
    -he game manager will then start the shooting game.

20. The player will be able to change parameters about the game.
    The game manager will implement the game settings that the user chooses from the settings menu.
    The settings will be used to change how the NavigationGame and ShootingGame will be played such as size area, difficulty, and other settings.

21. The player will be able to exit the application.
    The GameManager will be in charge of being able to exit the application.

Template created by G. Walton (GWalton@mail.ucf.edu) on March 28, 1999.
Last modified by Sarah Applegate on October 19, 2011.