Unknown Players

Zombie Smash

Andrew Baca
Muhammad Mohsin
Ahmed Mohamed

Description of Our Software

We are currently in the process of developing a 3-D infinite runner style game, where the user will control a car moving at a constant velocity down a straight road while avoiding hazardous obstacles, and trying to hit zombies to achieve a high score.

The scoring Mechanism will be based off a combination of how much time has elapsed since the game has started and how many zombies have been hit by the user controlled car.

Score = Zombies * 100 + TimeElapsed * 10,
Where the score will be displayed on a HUD, and in the game over display

The game will also automatically end once the user controlled vehicle makes contact with one of the numerous hazardous obstacles, and from the game over menu, the user will be able to customize the options, and start a new game session.

The main menu and the pause menu will consist of the same functionality

- an options menu consisting of game pace and sound options
- a GUI command to start gameplay, or resume gameplay

We decided to cut out some of the the additional features due to time (such as customizable cars and random perks), to really focus on the optimization of how the basic system runs

Key Architectural Drivers

Some of the Key Architectural Drivers that helped us decide the ideal Architecture Style included:

The fact that out system our system relies heavily object oriented programming tactics, where there is certain c-sharp scripts are attached to just about every different object, and these scripts have certain methods pertaining to the object.

Basically none of the Classes of objects will be working by themselves, they all rely a great deal on interactions with each other, and most of the methods from a certain object class will change the state of another type of object class based on events that occur.

The Architectural Style we chose to pick will rely heavily on certain components waiting on events to occur, and new events will be “triggered” once these events occur that certain components are waiting on…..

Our Ideal Architectural Candidates

Client-Server

Why:

Client components can subscribe to services of other object in game such as powerUps, obstacles, and zombies (these would act as a server in our system) who offer services to the car in return (speed up, destroy, increment score)

Drawbacks:

We need mostly all the components to act as both a client and a server, as well as an architecture that is more event driven

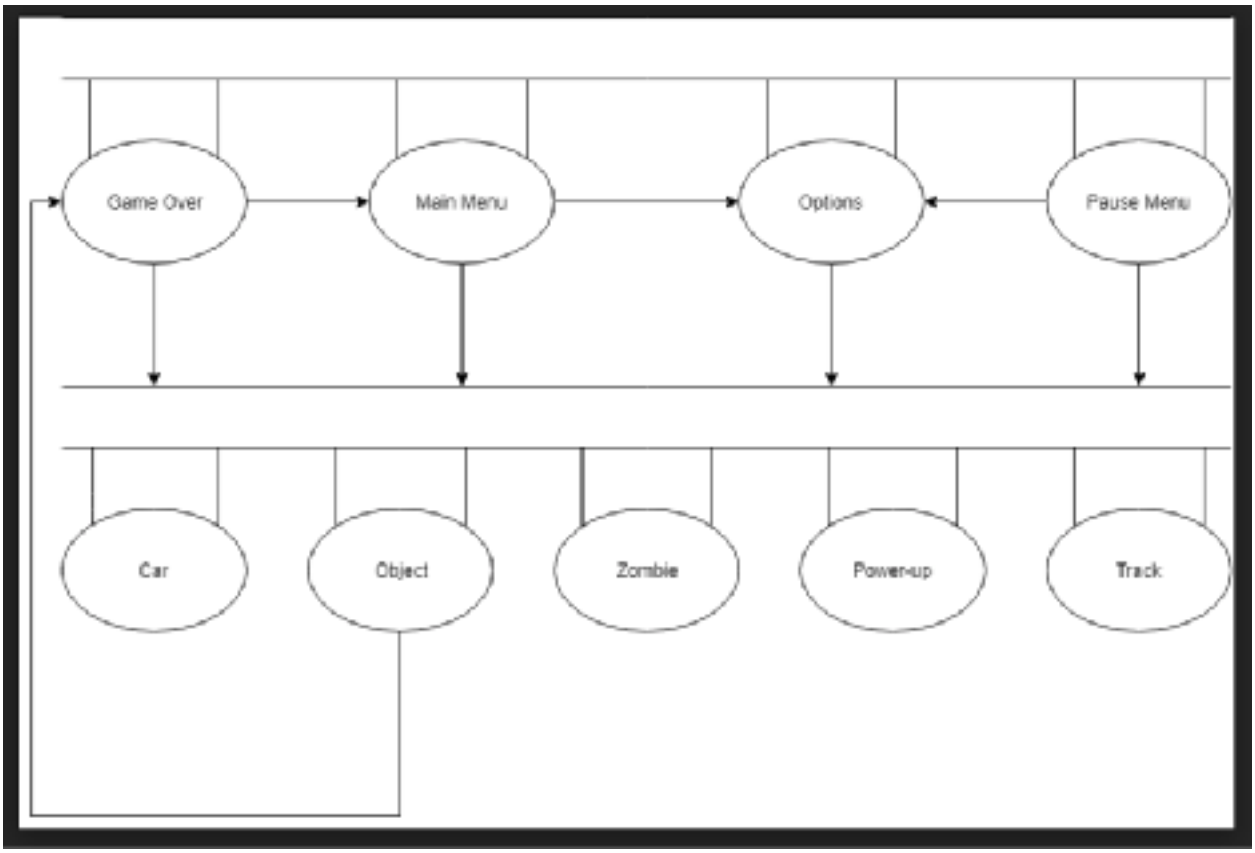
Publish Subscribe

Why:

Objects have the capacity to broadcast (act as a server) and react (act as a client) to events based on their interactions with other objects

This is the case for mostly all the objects in the gameplay portion of our software (except the track, which will exclusively serve as a client). For instance, all the objects in our game subscribe to the car component, waiting till an action (publishing) takes place. Once the car component has announced that one of these actions have taken place, events will be triggered based on the actions based on the other components that subscribed to it.

Our Architecture



Conclusion

We are going with Publish-Subscribe as the main architectural style of our gameplay portion of our game. It is ideal because the main concept of our design is that we have events be triggered when components come into contact with another component.

One risk is that the publish subscribe method can get tricky to debug and test when there are a multitude of objects subscribing to other objects an publishing to other objects (think of a tangly spiderweb)

We are still considering whether to use publish subscribe for the menus (main menu, pause menu, game over)
Any suggestions would be appreciated!!!