CIS564: Game Design and Development

Assignment #2: Deconstructing a Classic

Part A: Unity Tutorial **– Completed by: Monday, June 2, 2014**

Part B: Tech Spec **- Due: Tues., June 3, 2014**
(please submit to Canvas by midnight)

Part C: Base Gameplay **- Due: Monday, June 9, 2014**
(please submit to Canvas by 12noon)

Part D: Enhanced Gameplay **- Due: Wed, June 11, 2014**
(please submit to Canvas by 12noon)

Assignment Overview

In Part A of this assignment you need to complete the Unity Tutorial (available on Canvas). In Part B you need to analyze one of the classic Atari-age games below (i.e. Asteroids, Space Invaders, Frogger, Breakout or Missile Command), reverse engineer its gameplay, then write a design spec that can be used to reconstruct it. In addition to listing the game design goals and objectives, challenges and conflict, constraints and boundaries, win/loss conditions also include technical specs describing the required software data structures and classes, game play mechanics, algorithms and logic, and 2D/3D art assets required for the implementation. In Part C you will implement the spec generated in Part B using the Unity Game Engine. In Part D part you will extend/enhance/improve the game code developed in Part C to create new gameplay that is significantly different from the original game

Game Descriptions and Links

Asteroids:

In Asteroids, the player controls a single space ship in the middle of an asteroid field. At random intervals, enemy alien ships fly into view firing at the player. A level is cleared when all of the asteroids in the field are destroyed. A player loses a life when hit by an asteroid or hit by enemy fire.

http://www.neave.com/games/asteroids/
http://www.atariage.com/software_page.html?SoftwareLabelID=675

Space Invaders:

A single ship must hold off an approaching armada of alien enemies. Traditionally, the aliens move as a group downward in a serpentine pattern, firing down at the player's ship as they move. Each time an enemy ship is destroyed, the faster the enemy mass moves. A level is cleared when all of the enemy ships are destroyed. A player ship is destroyed either by being shot the enemy aliens or by having an enemy ship reach the bottom of the screen.

http://www.tripletsandus.com/80s/80s_games/invaders.htm
http://www.atariage.com/software_page.html?SoftwareLabelID=662

Frogger:

In Frogger, the player assumes the role of a frog that must make it from the bottom of the screen to the top of the screen alive in a fixed amount of time. To do this, the player must cross a street avoiding traffic of differing speeds and cross a river on moving logs and the backs of turtles. A level is cleared when each of the exit points at the top of the screen is filled with a frog. A player loses when he is hit by a car, falls into water, is carried pass the limits of the screen by on a log/turtle, or runs out of time.

http://www.tripletsandus.com/80s/80s_games/frogger.htm
http://www.atariage.com/software_page.html?SoftwareLabelID=621

Breakout:

In Breakout, the player controls a paddle at the bottom of the screen. At the start of the level a ball is launched from the paddle upwards towards the breakable block above. The ball ricochets off all walls, blocks and the paddle. When the ball hits a block, that block is destroyed. A level is cleared when all of blocks are destroyed. A player loses a life when the ball hits the bottom of the screen without being deflected back by the player.

http://www.tripletsandus.com/80s/80s_games/superbreakout.htm
http://www.atariage.com/software_page.html?SoftwareLabelID=668

Missile Command:

In Missile Command, the player defends a town from incoming enemy missiles. The player controls a cursor that tells where a player missile will detonate. Fire can come from any one of 3 base stations located on the bottom of the screen. All bases can be fired independently and have limited resources. A level is cleared when a set number of missiles are destroyed. The game is over when all player cities have been destroyed.

http://www.tripletsandus.com/80s/80s_games/missilecommand.htm
http://www.atariage.com/screenshot_page.html?SoftwareLabelID=618

Part A – Unity Tutorial

This tutorial will teach you the fundamentals of using the Unity3D engine and get you started on a shooting game based on Asteroids. The Unity Tutorial can be downloaded from the CIS564 Canvas site from the Assignment#2 Folder.

Part B - Technical Specs

Part B of the assignment is to create a technical specification for the game you have selected that will permit easy implementation in Part C. The technical specification should include the following information:

1. **GAME DESIGN:** Describe the following for the selected game: player goals and objectives, challenges and conflict, constraints and boundaries, resources, detailed description of the rules, including win/loss conditions.

2. **SCENE DESCRIPTIONS:** This section should describe all the scenes in your game. Example scenes in your game include: game intro, game menu, game level(s), win/lose scene, in-game menu, etc. For each scene you must provide the following information:
    a. Description: Brief description of scene layout and purpose.
    b. Game Object List: Enumeration of all game objects in the scene. Game objects include cameras, player avatars, non-player characters, world objects, abstract entities such as object managers, win/loss triggers etc.
    c. Player Input Specification: A list of all player inputs and the mapping to what event it triggers in the game. The events will correspond to methods which belong to components attached to specific game objects in the scene (see object description below).

3. **SCENE CONNECTIVITY:** A connectivity graph illustrating how control is transferred from one scene to another.

4. **OBJECT DESCRIPTIONS:** This section should describe all the objects/prefabs that will be used in the game. For each object, the following information must be provided:
    a. Assets: A list of all assets
        i. Models and geometry
        ii. Textures
        iii. Sound assets
        iv. Animations
        v. Shaders (custom or standard)
    b. Standard Components: A list of all standard components (transforms, meshes, colliders, rigid bodies, etc) that will be attached to each of your game objects.
    c. Custom Components: A list of all custom created components that will be attached to each of the game objects. Custom components encapsulate the data structures and functionality associated with each game object. Each custom component must in turn describe:
        i. Data members
        ii. Methods (manually invoked in program)
        iii. Triggers (automatically invoked by game engine based on certain events)

5. **INTER-OBJECT COMMUNICATION:** Objects may communicate by accessing each other's public data members, invoke public methods of other objects, send messages, or use triggers. This section should describe which objects in the scene communicate with one another and the details of the inter-object communication.

If you chose the asteroids game, features not included in the Unity Tutorial that need to be spec'ed out in Part B and implemented in Part C include:

- Asteroid velocity on spawn and movement
  - the Asteroids in your current implementation are kind of dumb: they just sit around on one spot when they are spawned! Give the Asteroids a random velocity and speed when they are spawned. Refer to the Bullet spawning code for simple velocity and speed, and the Asteroid spawning code for how to get Random numbers.
- Asteroid chunks
  - rather than having the Asteroids die entirely whenever they are hit, make the Asteroids release smaller Asteroids when they are hit (but make sure that there is a limit to how small it can get, or else the game will be impossible!)
- Wrap-around screen
  - whenever entities such as the Ship, Bullets, Asteroids, etc. cross one of the screen edges, they should come out the opposing screen edge. Two easy ways to implement this: create GameObjects at the screen boundaries that do collision detection, or add a check to each entities' X and Z coordinates in their Update() method.
- "levels"
  - in Asteroids, a new "level" occurs after the player has destroyed every Asteroid in the Scene. Currently, the Global script spawns Asteroids at fixed time intervals. Modify it so that it tracks how many Asteroids are currently in the Scene, and spawns more Asteroids once they are all destroyed. If you aren't sure how many to spawn at each "level", play Asteroids and find out.
- lives
  - right now, when Bullets contact Asteroids, the Asteroids are destroyed. However, when Asteroids contact the player's Ship, the player should die. The player has a limited number of lives. Play Asteroids and see how it handles the player respawning: is it ever the case that the player can spawn on top of an Asteroid? If so, are any mechanics in place to prevent the player from dying instantly upon respawn?
- UFOs!
  - every few seconds, spawn a UFO from the left or right side of the screen
  - make the UFO fly across the screen sinusoidally
  - the UFO shoots a projectile every few seconds that can damage the player or the asteroids
- limited number of Bullets that the player can fire
  - in games with shooting, there are two classic ways to limit how many times the player can fire: one is to use a cooldown timer after every shot has been fired, and the other is to track the number of Bullets that are currently alive, and refuse to create more if the maximum number is reached. Play Asteroids and find out which it uses. Try implementing both and see which feels better.
- High Score Screen
  - after the player has lost all lives, they should be taken to a high score screen where they can enter their name and see the current top scores.

Part C – Game Implementation

1. Implement your Part B spec using the Unity 3D game engine. Your game **MUST** be implemented in 3D (proxies such as cubes, spheres, tetrahedrons, etc. for the 3D geometric models are fine, although custom models or models obtained online at sites such as turbosquid.com will produce a more polished look). All games must have, in addition to the features described above in Part B, a 3D effect (for example: first person view, camera shakes on impact, external controllable camera, etc.) that was **not** part of the original game and which leads to some interesting/novel game play.

3. At a minimum, your implementation must be capable of demonstrating the basic game play of the original Atari-age game, including:
    a) A win and loss state
    b) Sound effects
    c) Score display (2D)
    d) A 3D mode for demonstrating the 3D gameplay/effect.

3. Please submit your Unity project along with an executable version of the game and a video demo (recorded using either Camtasia or Fraps) to the CIS564 Blackboard site. Please use a video format such as MOV, WMV, MPEG4 when creating your video (AVI files are not acceptable).

Part D – Game Enhancement

Now that you've gotten a working core for your Atari-age game, your task in this part of the assignment is to take that code and **extend/enhance/improve it in order to create new gameplay that is significantly different from the original game**.

In this part of the assignment you must do the following:

1. Create **at least 2 new game rules** that makes significant changes to the goals/objectives, boundaries/constraints or challenge/conflict of the original Asteriods game

2. Create **at least 1 new significant resource**
    - Where the new resouce must have some scripted interaction with other game elements, some kind of audio, and some kind of visual representation

3. Create **a new scoring system** for your game that reflects the new gameplay features
    - it cannot just be an adjustment of the values of your existing scoring system (and you are not allowed to simply have no scoring system)

4. Provide a brief written description documenting your new features in a Readme file and submit this along with your updated Unity project code, game executable and an associated video demo to the associated CIS564 Canvas folder **by 12noon on the due date**.