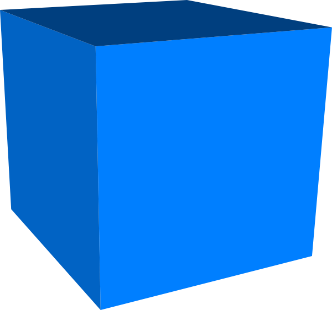
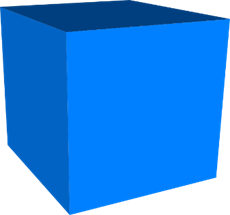
CubeBoids!

Simulating Flocking Algorithm

Technical Design Document

(With Complexity Analysis)



Student: Ohmeko Ocampo

Professor: Steve Price

Date: 2 April 2017

SDSU Fall 2017

|  |  |
| --- | --- |
| Table of Contents  [1 Executive Summary](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)   1. Game Overview 2. Technical Summary   [2 Equipment](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)   1. Hardware 2. Software   [3 Evaluation](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)   1. Game Engine 2. Target Platform   [4](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel) Scheduling   1. Development Plan 2. Milestones 3. Updates, Maintenance & DLCs   [5 Work Environment](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)   1. Remote Collaboration   [6 Levels](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)   1. Level 1 (World) 2. Asset List   7 Complexity Analysis | Game Development Team Members    PRODUCER  Ohmeko Ocampo    GAME DESIGNERS  Ohmeko Ocampo  PROGRAMMERS  Ohmeko Ocampo  TECHNICAL ARTISTS  Ohmeko Ocampo |

|  |
| --- |
|  |

# 1 Executive Summary

Game Overview

CubeBoids is a simulation of the Flocking Algorithm described in the book Unity AI Game Programming – Second Edition. The algorithm is used to describe the flocking and school behavior animals such as bird and fish. In this implementation of the algorithm, the birds are represented by the Unity3D cube primitive hence the name.

Technical Summary

CubeBoids! will be developed in approximately the whole SDSU year using the Unity Game Engine. I mentioned whole SDSU year because I will be going back to this program and refining it even after class has ended. There was no real 3D asset creation in this game as all the characters in it are just Unity primitive 3D shapes. The total production cost of the game will not exceed more than $2000 dollars. This game was built as a demonstration in learning how the Flocking Algorithm works in the Unity Environment.

The game will be deployed for the Windows 10 OS environment.

# 2 Equipment

Hardware

I used a MSI GE72 6QF Apache Pro for putting the game together and play testing the game on Unity.

\*Note\*: (We already have this equipment but just for the purpose of learning how to do a TDD I just put this in here anyway)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Product | Task | Cost | Quantity | Total |
| MSI 6QF Apache Pro | Play Testing  Programming | $1,200 | 1 | $1,200 |
|  |  |  |  | Total: $1,200 |

Software

No art software was used in the development of “CubeBoids!” so no expense was spent in art creation, however I had to use Microsoft Office for creating the GDD and TDD documents.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Product | Task | Cost | Quantity | Total |
| Microsoft Office | Create the GDD and TDD documents | $120 | 1 | $120 |
| Unity (Free Edition) 5.5.0f | Creation of Game | $0 |  |  |
|  |  |  |  | Total:$120 |

# 3 Evaluation

Game Engine

The game engine utilized for the development of CubeBoids! is Unity. We choose development in Unity because this project is for CS 596 Advance 3D Game Programming which used Unity for game development. Unity can make a 3D game with ease, we can make it highly optimized and beautiful, and we can deploy it with a click to multiple platforms. In addition we can use Unity’s extensive libraries and integrated services to speed up our development process such as the prefab, optimize our game, connect with an audience, and achieve success.

Target

CubeBoids will be deployed to Windows 10 since it was the system I developed CubeBoids on.

# 4 Scheduling

Development Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Product | April 2 | April 3 | April 4 - 5 | April 6 |
| 2D Art | n/a | n/a | n/a | n/a |
| 3D Art | Just used a sphere for the leader, and cube for the cubeboid. |  |  |  |
| Scripting | Agent Script and AgentConfig  Alignment()  Separation()  Cohesion() | World Script  Instantiate Boids and Keep them bounded. | Credit Script  Menu Script | Pause Menu Script |
| Audio | BirdSoundsForKids.mp3  For Menu, Levels, and Credit Scene. |  |  |  |

Milestones

|  |  |
| --- | --- |
| 4 April 2017 | Finished Programming the CubeBoid game. |
| 6 April 2017 | Finished Documentation: GDD, TDD, READ.md |

# 5 Work Environment

I worked from my bedroom, SDSU Library, and my living room.

# 6 Levels

CubeBoids will consist of two scenes that respectively contain a different flocking mode either “Lazy Flight” or “Follow the Leader”. The level will have generated CubeBoids and be bounded so that Cubeboids don’t fly everywhere.

Asset List

|  |  |
| --- | --- |
| Players | Cube Primitive as CubeBoid |
| Environment | Plain Blue Background that’s bounded. |

# 7 Complexity Analysis

Looking at all of my scripts, the most process heavy instruction would definitely be the instantiation process of the CubeBoids in the World.cs script as shown below in Figure 1.

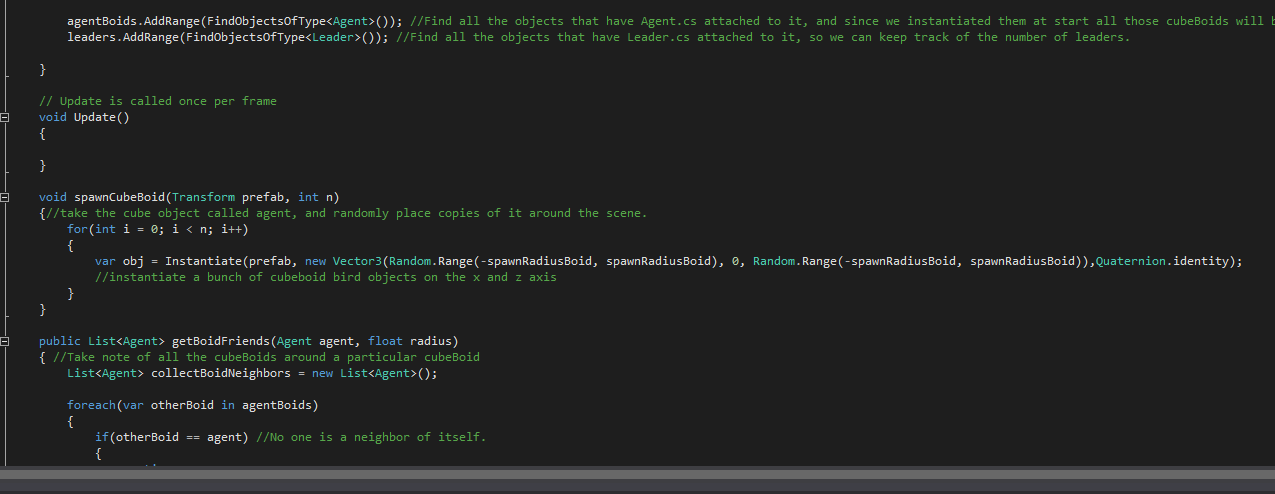


Figure 1. spawnCubeBoid() function instantiating Boids.

I noticed that around 300 CubeBoids, my flocking levels especially the “Follow The Leader” mode was especially slow. Since there are always n number of CubeBoids there should be n things always to process so the complexity class should be O(n).

However, that would probably be the naïve way of thinking about the complexity of this algorithm considering that the code uses a lot of vector math to help each boid calculate how much “effort” it should put forth either in cohesion, separation, or alignment. This is not to mention the other method I added which is the followLeaders() function which further complicates the decision for the CubeBoid. The follow the leader method is shown in Figure 2 below.

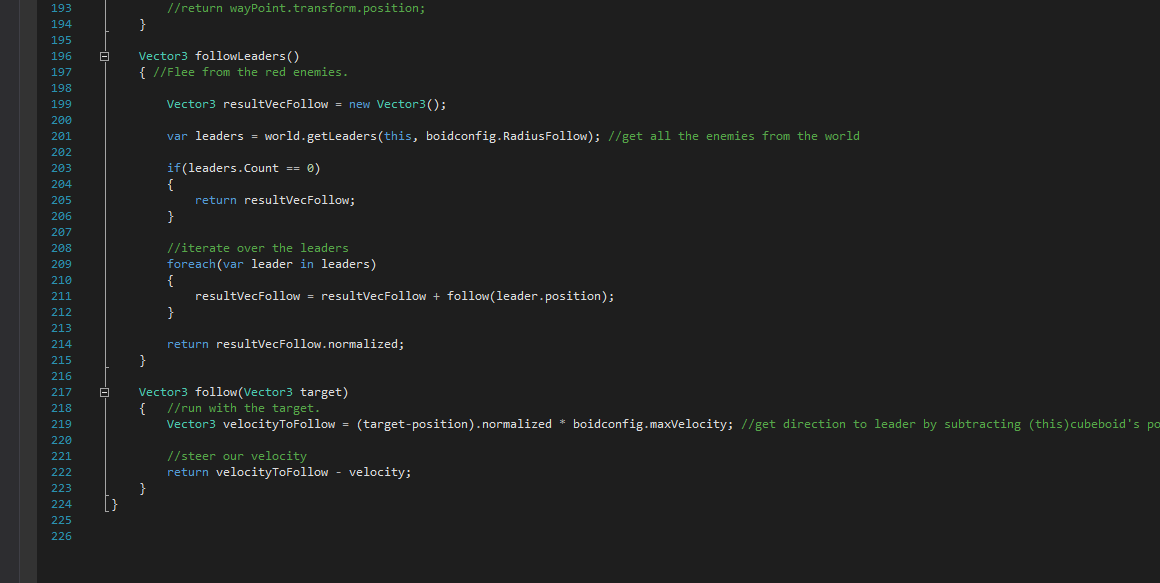


Figure 2. followLeaders() and helper function follow() for follow the leader tendency.

Giving a little bit more thought into the complexity of this program we can see that we would have to think of the complexity in terms of the complexity when you add two matrices together. Thinking about a N row matrix and another N row matrix, the complexity of this program would seem not to be O(n). Thinking about matrix multiplication we know that for each element in n row vector matrix you add each element by element to another n row vector matrix element by element.

Now the complexity of adding both of these n row vector matrices would be O(n^2) since you would effectively be visiting each element in the row vector exactly once each way. Since this complexity class is more complex then O(n) the final classification I’d give to the flocking algorithm is:

**Flocking Algorithm** belongs to O(n^2).