# Collision Avoidance with Unity3d

Jassiem Ifill

September 12, 2013

**Abstract**

The primary goal of the research presented in this paper is to achieve natural crowd simulation and collision avoidance within the Unity3d game engine as it is becoming a very popular choice of development software.

Throughout this summer, a section of our research group was focused on using the Unity3d game engine and we attempted to use two different methods of achieving Collision Avoidance to see if they were compatible and feasible within the Unity3d engine. As such, this paper builds upon a previous Predictive Force Collision Avoidance method and use a variation of it in order to achieve a working form of Collision Avoidance within the Unity3d engine.

The final result of this project was a fully operational windows application that allows the user to choose between two different algorithms for collision avoidance. After that, the user is then able to use this version of the Predictive Force Collision Avoidance Model to either: play a simulation containing agents with working collision avoidance with as many agents as desired, render a simulation from a file, or export a simulation with a specified amount of agents and length to a file.

## 1 Introduction

Within this research, we are focusing on implementing various methods of Collision Avoidance within the Unity3d game engine. We have decided to do this because Unity3d has become a very popular game engine and many developers are now switching over to the Unity3d game engine as it presents a user friendly development environment with various resources. Moreover, Unity3d has three different languages that can be used for scripting and they are C-Sharp, JavaScript, Boo (Python). Likewise, Unity3d also has its own animation system whilst still allowing users to import models from other rendering software such as Maya or Blender. As such, my co worker David and I started working with the Unity3d game engine so that we could start to figure out how some parts of the engine work. We initially started with simple tutorials and afterwards, we created small goals to work towards whilst building up to our final project at the same time. Throughout the summer, I also used a variety of methods and algorithms in order to move towards the final project, and to solve the various problems that we encountered.

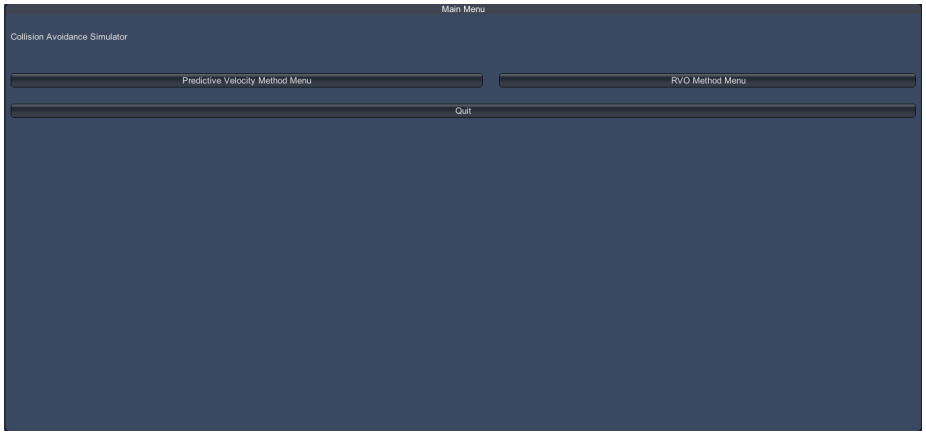


Figure 1: Primary Graphical User Interface (GUI) for the application

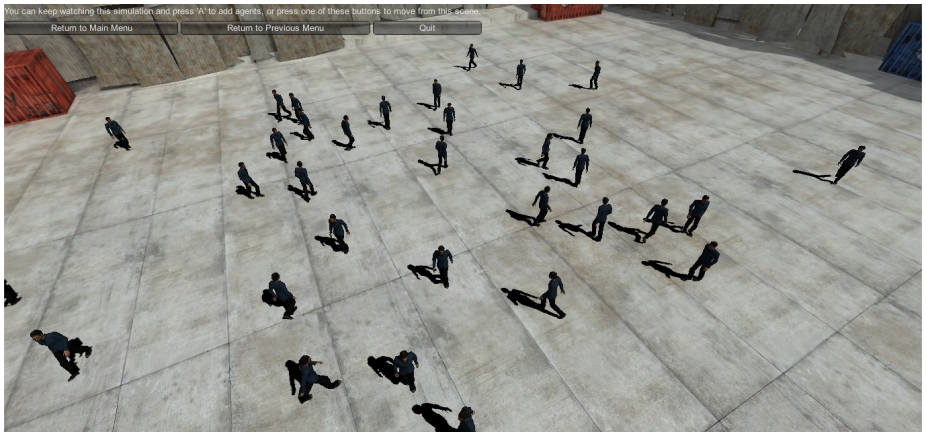


Figure 2: Overhead View of Simulation


Figure 3: Chase View of Simulation

Additionally, we worked on implementing the field of Crowd Simulation and Collision Avoidance within the Unity3d game engine as Crowd Simulation is very important in todays society. One of the reasons for this is that the focus on safe and efficient crowd management is becoming more essential due to the population of the world increasing, as well as urbanization. As such, Crowd Simulations allows one to analyze and even optimize crowd behavior in a given situation. Possible uses of Crowd Simulation include, but are not limited to, evaluating structures throughout their building process, creating and testing the efficiency of evacuation plans, detecting safety risks and security risks, comparing multiple scenarios at once, and determining the most efficient size of a structure in order to achieve a specific goal. Collision Avoidance is a part of Crowd Simulation and virtualization that deals with having agents avoid collision with each other, or certain obstacles. Collision Avoidance is used within many different types of Crowd Simulation and even other types of simulation and virtualization. Within Crowd Simulation, Collision Avoidance is important as it is needed to properly simulate crowds in a lifelike and natural manner. Without Collision Avoidance, one would not be able to use Crowd Simulation to test the safety and efficiency of certain structures as the agents would not behave like normal humans. Moreover, Collision Avoidance is also used in a variety of games, and other virtual simulations.

## 2 Related and Prior Work

The subject area of Collision Avoidance within Crowd Simulation has been touched by a plethora of previous works in the past. One of these works is explained in the "Reciprocal n-body Collision Avoidance" paper by Berg et al. [6]. This paper addressed the issue of collision avoidance within Crowd Simulation by posing a geometric way of achieving Collision Avoidance agents in a small clustered area. To achieve this, Berg et al used a velocity based method in which the agents would create a half-plane of, or a space of, velocities that will allow the agent to avoid collision. Then, the agent would select the optimal velocity using a form of "linear programming".

Similarly, the work of Karamouzas et al.[3] suggested a predictive method of avoiding collisions for pedestrians. Moreover, the method that Karamouzas et al. is what the method used in this paper is derived from. Whilst using the method that was created in this work, the agents would predict possible future collisions and make moves to avoid them. Karamouzas et al. noted that this method provides smooth collision avoidance behavior along with visually satisfying simulations in their experiments. Furthermore, Karamouzas et al. goes on to utilize the predictive force base method that they created to avoid collision, and they also have each agent predict collision by having them find their future positions and detect whether it will be within the "personal space"of another agent.

Also, the work of Guy et al.[1] presented a solution for multiple agent simulations that was based on velocity obstacles. Moreover, the algorithm produced

by this work was said to work well in dense simulation scenarios as well as being able to perform faster than algorithms that came before it. The Clear Path approach also built off of the work of Berg et al.[7] which used RVO or reciprocal velocity obstacles for the navigation of multiple agents.

More Important works in the field of collision avoidance include the works of Reynolds [4][5], and the work of Helbing et al.[2].

The work of Reynolds[5] raised the problem of simulating a flock of birds, or any other group of animals, moving naturally. Reynolds[5] acknowledged how difficult it would be to scrip each bird in a flock individually, and he also noted that a flock scripted in this manner would be hard to edit. As such, Reynolds posed the notion that a flock was simply the result of the behavior of individual birds when they interacted. Hence, Reynolds believed that in order to simulate a flock, all that would be needed was to simulate an individual bird with the aforementioned behaviors and create multiple instances of that bird. To build this flock, Reynolds started with a bird object or boid that supported flight. Afterwards, behaviors that lead to simulated flocking were added. These behaviors consisted of Collision Avoidance, Velocity Matching, and Flock Centering. The Collision Avoidance feature of the boid models are said to be predictive, and they also establish the minimum required separation distance of each boid model from another. Furthermore, Reynolds called the results of these behaviors, behavioral urges and implemented a navigation module in the brain of each boid to manage these urges. This work is important in the field of Collision Avoidance and Crowd Simulation because it demonstrates how collision avoidance can be combined with many other methods to achieve a natural simulation, among other reasons.

Another work, also by Reynolds [4], addresses the issue of having virtual characters moving around their world in a life like and natural manner. Such a matter is very important in video games, as having natural AI behavior is essential in creating a realistic environment. In this paper, Reynolds focused on making autonomous agents that were life like and also had the ability to improvise in certain situations. Additionally, Reynolds divided the behavior of autonomous characters into three layers known as Action Selection, Steering, and Locomotion. It was also mentioned that the behavioral hierarchy mentioned in this work is applicable to motion behaviors and that it is not well suited for other non-motion related behaviors. Reynolds work focuses on the steering layer of his behavioral hierarchy, and it also presents a version of obstacle avoidance within the steering behaviors section. Within Reynolds version of obstacle avoidance, the character and the obstacle are both treated as spheres in theory. Additionally, the character keeps an imaginary cylinder in front of it in order to represent its required free space. This cylinder faces forward from the character and extends a certain distance from the center of a character based on the characters speed and agility. Reynolds obstacle avoidance feature looks at all objects and determines if they intersect with the cylinder. Moreover, in Reynolds obstacle avoidance behavior, if the distance between the character and the obstacle is greater than the sum of their radii, then Reynolds behavior will determine that there is no potential collision. Also, the obstacle avoidance

either returns a steering or avoidance value, or a special value to signal if there is no collision. Reynolds also explains a plethora of other steering behaviors for autonomous characters such as seek, pursuit, flee, and many others.

Another instrumental work in the field of Crowd Simulation and Collision Avoidance was the work of Helbing et al.(reference here). The work of Helbing et al. attempted to address the issue of using pedestrian simulations to help solve real world problems. The issues that Helbing et al. addresses include predicting how pedestrians will act upon panicking or being jammed in a certain structure. Also, the primary focus of this work is modeling the occurrence of escape panic. Within this work, Helbing et al. introduced a psychological tendency of two pedestrians to stay away from each other. Moreover, they implemented this tendency as a repulsive force. This force acts as a version of collision avoidance for the pedestrians. Additionally, if two pedestrians happen to touch, Helbeing et al. also added a body force and a sliding friction force to simulate how panicking pedestrians would act if they are touching another pedestrian.

# 3 Methods

After we completed simple tutorials, which included instructions on how to make a 2d side scroller runner game, and a moving clock, we then started to utilize skills that we had learned from the tutorial in order to take one of the first steps towards our goal. As such, we began to create cubes on the screen that would randomly move to a goal, and then choose a new goal.

To do this, we created cube objects in a similar manner to what we did in the tutorial and we initially utilized the "'MoveTowards" method of Unity's "Vector3" class which moves an object towards a point on a plane. Additionally, I generated a random goal by using the random generator class of the Unity3d engine, and made that the point that each cube would move to. Furthermore, I modularized the process of getting a random goal by creating a function to do that task by itself and this function was called every time a cube reached it's goal. Afterwards, we proceeded to create "prefabs" that were instances of each object as we had learned to from the tutorial. Then, I created a function that would instantiate each prefab upon the press of a button.

After this initial step, we started working on implementing some form of collision avoidance now that I had objects moving on the screen. Our first idea was to see if there were any common methods of achieving collision avoidance with Unity3d. Upon search, I heard of a method called "raycasting" which consisted of shooting a ray in a certain direction. This method consisted of using the "'RayCast" method of Unity's "Physics" class in order to shoot the ray. After shooting the ray in a certain direction, I would check if the ray came into contact with another object in order to detect collision. However, the only problem with this is that the objects would simply move to a random position when they were about to collide and they would never actually reach their intended goal. Ensuing this, we decided to use a different algorithm that consisted of each object finding the closest object to it, and then having that object focus on moving

away from the closest object instead of moving towards the goal. In other words, moving away from the closest object would take priority over moving towards the goal depending on how imminent the collision was. To do this, I created a vector pointing away from the closest object and a vector pointing towards the goal. Following this, I normalized these vectors (which gave them a magnitude of 1) and used them as the different parts that would make up any given object's velocity. In this first prototype, I only implemented the avoidance portion of the velocity if the closest object was less than 2 units away. Furthermore, I would simply add the velocity to the objects position every frame instead of using the MoveTowards" function as before. The reason behind this change is that manually editing the position gave me greater control over the behavior of each object. Although this method proved to successfully produce collision avoidance, the movement of the objects was very choppy, jerky, and sudden and hence, it could not be used to simulate human movement well. As such, we once again revised the current method of collision avoidance to anticipate collision similar to the way a normal human would, instead of waiting for an object to get very close.

The new algorithm that I started to implement is a variation of the Predictive Force Model that was made by Dr. Ioannis Karamouzas [3] . This algorithm uses the formula for a line intersecting a circle as it's base, and it is derived from there as shown below.

$$(P - C)^2 = R^2 \tag{1}$$

$$((Po + (t * V)) - C)^2 = R^2 \tag{2}$$

$$(Po + (t * V) - C)^2 = R^2 \tag{3}$$

$$((t * V) + (Po - C))^2 = R^2 \tag{4}$$

$$((t * V) + (Po - C))^2 - R^2 = 0 \tag{5}$$

$$(t * V)^2 + 2((t * V)(Po - C)) + ((Po - C)^2 - R^2) = 0 \tag{6}$$

$$(V^2) * t^2 + ((2V)(Po - C)) * t + ((Po - C)^2 - R^2) = 0 \tag{7}$$

$$(a) * t^2 + (b) * t + (c) = 0 \tag{8}$$

$$(a) = (V^2), (b) = ((2V)(Po - C)), (c) = ((Po - C)^2 - R^2) \tag{9}$$

The algorithm then used these values for a, b, and c and inserted these value in the quadratic formula to solve for the time in which the line would intersect

the circle. However, in reality, this would be the time that the two objects would collide with each other. This is achieved by treating the object in question as if it were they point, and the object that will potentially collide with it as if it were the circle, and giving it the combined radius of both objects. Moreover, the quadratic formula will then give out two times in which the point or object will intersect the circle, given that the determinant is positive. Likewise, if the times are both negative, that means that there was a collision in the past and that they will not collide in the future.

As such, in my code, I handled situations in which the determinant was negative, and both times were negative. Furthermore, I created a function in my code to implement this portion of the algorithm which returned the time of collision, regardless of the case, each frame. Also, If the determinant is negative or if both times are negative, a special negative value is returned from my getCollisionTime function for both of these cases. If neither of these cases occur, the lowest non negative time for collision is returned. After the getCollisionTime is called and returns a value in my movement function, the aforementioned special cases are handled if they occur. To elaborate, in these cases the object will have a portion of its velocity going towards the goal and a portion of its velocity going away from the closest object. Moreover, the magnitude of the second portion of the objects velocity is inversely proportional to how close it is to the nearest object. However, in the normal cases, the objects velocity will contain a component that points towards the goal, a component that points away from the closest object, and a third component, called the avoidance component, that allows the object to anticipate collision by moving away from its future point of collision. Moreover, the avoidance component of the velocity is a vector that points from the future position of the other object upon collision towards the future position of the object in question, so as to move away from the point after predicting a collision.

After applying these methods, the objects were successfully avoiding each other whilst walking around. Afterwards, I utilized 3D models instead of cubes or capsules as the objects in question. Following this, I used the Mecanim Animation system of the Unity3D game engine to create an animator controller for the 3D models. This animator controller would basically control the animation of each agent, and when the animation would play.

The last step after this was creating the GUI for the final project regarding this work. To create the GUI, Unitys GUI and GUILayout classes were used, along with the contained methods and properties. Moreover, the Window method of the GUI class was used to create the window, and the Button, CheckBox and TextField methods were used to further detail the GUI. Following this, certain parts of the code were linked to the event handlers for the buttons and textboxes so that the GUI would function as desired.

# 4 Discussion and Future Work

In this paper, we present a way to apply a variation of the Predictive Force Collision Avoidance Model within the Unity3d Engine. Moreover, this project allows the user to add agents upon button press, or to simulate a given amount of agents and export the simulation to a file. Additionally, another feature of the final program allows the user to render simulations from a file of a certain format. This allows users to see how certain simulations would be rendered in a specific amount of frames. Also, the final program of this paper also includes a variation of the RVO collision avoidance model that was produced by David Cherry, a fellow coworker in Dr. Stephen Guys lab.

There are many opportunities for future work regarding this program. One idea includes expanding this program into a web application so that users may simulate how agents would move in a given structure. Also, this would allow users to study how agents would behave under specific scenarios in any structure. Additionally, another idea could be to further develop this variation of the Predictive Force Collision Avoidance Model so that it would be more efficient similarly to the original Predictive Force Collision Avoidance Model created by Karamouzas et al.[3].

# 5 Acknowledgements

I'd like to thank Dr. Stephen J. Guy for directing the Applied Motion lab, for mentoring David and I throughout our research during the summer, and for helping us work through the problems that we encountered during our research. Id like to thank Dr. Ioannis Karamouzas for mentoring us throughout our research this summer and for helping us work through many problems that we encountered. Additionally, Id like to thank Julio Godoy, David Cherry, John Koenig, Devin Lange, and Bilal Kartal for being great co workers, and for being supportive throughout any problems that I encountered in my project.

# References

[1] Guy, S., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., and Dubey, P. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), pp. 177–187.

[2] Helbing, D., Farkas, I., and Vicsek, T. Simulating dynamical features of escape panic. *Nature 407*, 6803 (2000), 487–490.

[3] Karamouzas, I., Heil, P., van Beek, P., and Overmars, M. H. A predictive collision avoidance model for pedestrian simulation. In *Motion in Games*. Springer, 2009, pp. 41–52.

[4] Reynolds, C. Steering behaviors for autonomous characters. In *Game Developers Conference* (1999), pp. 763–782.

[5] Reynolds, C. W. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics 21*, 4 (1987), 24–34.

[6] van den Berg, J., Guy, S. J., Lin, M. C., and Manocha, D. Reciprocal n-body collision avoidance. In *International Symposium of Robotics Research* (2009), pp. 3–19.

[7] van den Berg, J., Lin, M., and Manocha, D. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation* (2008), pp. 1928–1935.