COMP 790-058 3D Game Engines
Project Report
Yue-Ling Wong (ylwong@cs.unc.edu)

Abstract

This project explores the use of the RVO2 (reciprocal velocity obstacle) library for the iPhone game development. Here, an iPhone game is developed using the RVO2 library, iPhone SDK, and Cocos2d game engine. The game scenario puts the player in a crowded theme park. The goal of the game is to visit all the attractions/rides in the least amount of time. The RVO2 library is used to control the crowd movement in the game. The RVO library is provided as source code in C. The game is implemented in Objective C and makes function calls to the RVO2 library. This report discusses the uses and modification of RVO2 made for the game. In addition, this report will also discuss the strategies used to simulate the crowd pattern in the leisure environment, in which the crowd does not always follow the shortest path to the end goals. The improvement on the frame rate using Kd-tree to reduce pairwise operations between agents and obstacles will also be discussed. The game is an isometric 2D game but can be developed as a 3D game using a 3D game engine.

1. Introduction

Crowd simulation simulates the movement of a large number of objects. Generally, obstacle avoidance and motion planning algorithms are used to determine the object's movement. In a crowd simulation that employs motion planning to seek the shortest path, the agents will follow a straight line whenever possible. For example, in the early stage of my project development, a straight run of the simulation testing with only three goals without obstacle showed the crowd movement in a straight line between goals (Figure 1). However, in the leisure environment, an agent's decision making of direction is not purely goal-driven but is based on stimuli and agent-to-agent communication [2]. Although it is not the goal of this project to simulate the *realistic* theme park goers' behaviors, this project investigates some strategies to simulate a random wandering crowd pattern in a leisure environment for games.

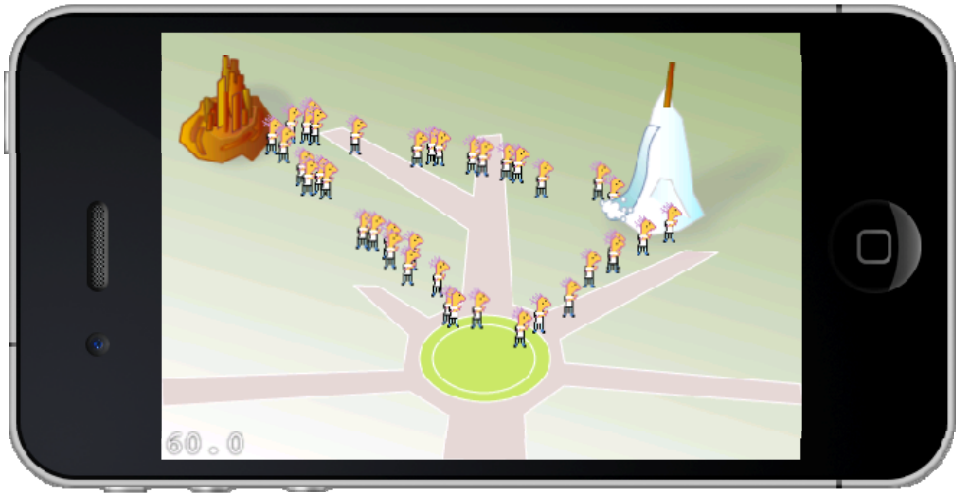

Figure 1. A screenshot of a straight run of the simulation test with RVO2 in the early stage of the project development with only three goals without any obstacles.

The RVO (reciprocal velocity obstacle) and RVO2 libraries, developed by the GAMMA group at UNC Chapel Hill [1], are libraries for crowd simulation. The libraries are based on an algorithm for interactive navigation and planning of large numbers of agents in two-dimensional environments. RVO2 has been

implemented in C++ and C#. An interactive simulation demo that simulates and renders a large crowd in real-time, and allows a user to direct the crowd interactively to exit an exhibition hall [3] has demonstrated the potential application of using crowd as a theme for computer games. The C implementation of RVO2 is currently under development and offers potential uses for iPhone game development. This project explores the use of the RVO2 library for the iPhone game development.

The game scenario of this project puts the player in a crowded theme park. The RVO2 library is used to control the crowd movement in the game. The game has a player character which is one of the agents of the crowd. Instead of the player directing the crowd, the player directs the player character to visit all the places in the theme park as quickly as possible.

In this report, I will use the terms *attractions* and *rides* interchangeably to refer to all types of the places that a guest in a theme park may visit. These may include rides, shows, and even food stands and restrooms. I will also use the term *player character* to refer to the agent that the player can assign a goal to. The player character is also an autonomous agent like the other agents of the crowd except that the player can assign a goal to the player character any time. The player can tap on the screen to change the player character's goal any time. When the player character has reached a goal, it will automatically select a random goal as the new goal.

2. Development Tools

The game is developed for the iOS4 using iPhone SDK and Cocos2d game engine. Cocos2d [4] is a free open source framework for building 2D games, demos, and other graphical/interactive applications. It has two integrated physics engines: Chipmunk and Box2d. It also allows easy generation of particle effects. Cocos2d has been implemented in Python and in Objective C for iPhone. Cocos2d for iPhone uses OpenGL ES to handle graphics. However, with theCocos2d API, you do not need to write openGL code. For example, Figure 2 shows a simple code snippet that puts a PNG image on screen.

```
CCSprite *player = [CCSprite spriteWithFile:@"kid.png"];
player.position = ccp(100, 250);
[self addChild:player];
```

Figure 2. Code snippet that puts an image named "kid.png" on screen at the location (100, 250).

3. Graphics and Sound

All the graphics and animations are created by me. The sound effects are either Windows system sounds or acquired from the Microsoft Office ClipArt. Some sounds are edited to trim off the silence at the beginning and the end of the file.

4. Gameplay Features

Figure 3 shows the full game map with 150 agents. The size of the map is 600 $\times$ 600 pixels. It is a scrollable map. Figure 4 shows a screenshot of the game. There are six rides and a restroom that an agent can visit. The six rides are as follows.

1. Haunted House
2. Triassic Period
3. Lochness Aquarium
4. Dizzie Carousel
5. Enchanted Canyon
6. Iceberg Adventure

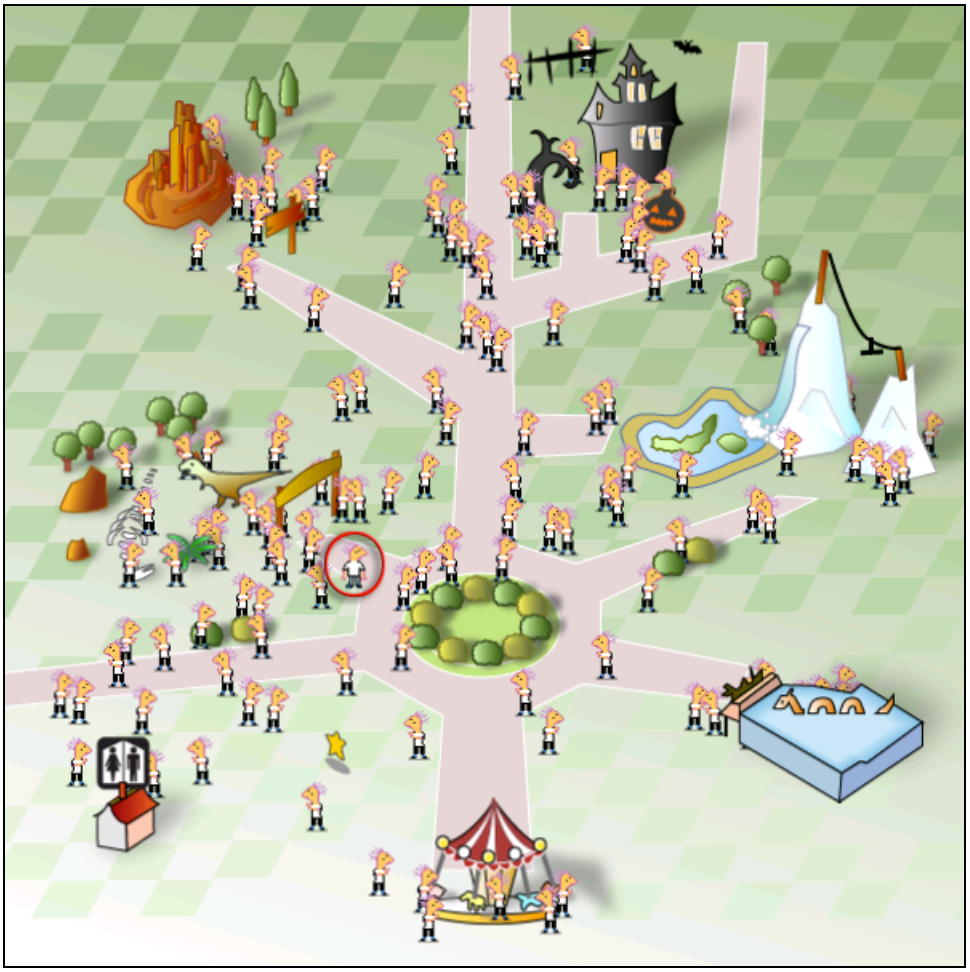

Figure 3. Full map (600 x 600 pixels) of the game. The player character is circled in red.

Figure 4. A screenshot of the game.

This section describes the gameplay features implemented in this game and the rationales behind them. These features intend to make the game more fun to play. One of the gameplay design goals is to create challenges for the player to solve and to keep the player interested. These challenges have to be within the player's ability to solve. Otherwise, the player will get frustrated and give up. The challenges should not be too easy and too repetitive or the player will get bored.

4.1 Player Character Blending in with the Crowd

The player character is intentionally designed to have a similar graphic as all the other agents, which share the same graphic. This creates a challenge similar to the popular "Where's Waldo" puzzles. Because the player character is constantly moving, the player has to keep a close eye on the player character--another challenge that may keep the player interested.

4.2 Providing Hints of the Player Character's Whereabouts

During the course of the project development and testing, I found that losing the player character is a little frustrating at times. I may want to give up if I could not find the player character within a few seconds. So I decided to provide hints for the player. I wanted the hint to blend in with the theme naturally. That is, the hint will not be something like a textbox popping up on screen telling the player where the player character is. I programmed the player character to jump up and down waving its arms (Figure 5) for several times every time the player taps on the screen to assign a new goal to the player character.

Figure 5. The image sequence of the player character jumping up waving its arms.

4.3 Player Character being an Autonomous Agent

In the early stage of the project development, the player character was programmed to stop and stays at the goal location, after reaching a goal, to wait for the player's next goal assignment. Later in the project development, I found that it is more fun to let the player character be an autonomous agent like all the other agents in the simulation. That is, after the player character has reached the goal assigned by the player, it will randomly select another goal. This mimics the behaviors of kids.

4.4 Adding "Magical" Feel Using Animation

When the player taps on screen, animated bubbles will continue coming out from the location. This was originally intended to create a "magical" feel of the game. However, I also have found that this actually serves as a visual element to let the player know whether the player character has reached the tapped location or not. For example, I may tap on the screen and the bubbles keep coming out. Then, when I do not see the player character coming to my tapped location within two seconds, I may get distracted and start scrolling the map. Few seconds later, I may scroll the map back to the tapped location to check on the player character. If the bubbles disappear, I know that the player character has already reached my assigned goal and now is going to another goal. I may tap on the screen again to try to get the player character back. If the bubbles are still there, then the player character may have been going through a crowded area. In this case, I may want to scroll the map to find the player character and try to direct it to a less crowded route even if it will be a longer route.

4.5 Animation of Rides as an "Reward" and For Keeping Track of the Visited Rides

When the player character has reached the entrance of a ride, the ride will start to play an animation. For example, the haunted house will have the lights flickering and the bat hovering over the house like crazy. Table 1 lists the animation of each ride in the game. The fun animation serves as a motivation for the player to reach the ride. In addition, the animation serves as a visual "checkmark" to help the player keep track of which rides have been visited. In addition to the animation, the first time when a ride has been visited successfully, an alert prompt (Figure 6) will appear with a "ta-da" sound effect to let the player know.

Attractions	Animation after the ride has been visited
Haunted House	The lights in the house flicker and the bat hovers over the house like crazy.
Triassic Period	The T-rex does the howling motion.
Lochness Aquarium	The lake creature goes in and out of the water surface.
Dizzie Carousel	The carousel spins like crazy.
Enchanted Canyon	Colorful stars come out of the canyon and spin around the canyon.
Iceberg Adventure	It snows at the iceberg area.

Table 1. Animations associated with the attractions in the game.

Figure 6. After the player character has reached an attraction, a prompt to pop up to give the player a confirmation.

4.6 Collecting the Spinning Star

In general, kids (many adults do too) have a desire to collect things, such as Pokémon cards, shells, charms, and stuffed animals. This desire is not lost in the computer game environment [6]. Here, I added a spinning star, which randomly appears in the theme park. The player character can collect it by going near it. A magical chime "bing" sound will play. Collecting the star is a diversion of the game; the player character needs to take a detour to get the star. This diversion is a game challenge designed for the player--weighing racing to the ride against the desire to collect a star that has no consequence to the game goal. However, the player would probably enjoy such distraction if the player has a desire to collect objects. Of course, the number of spinning stars collected will be kept track--displayed on top of the screen.

4.7 Timer and Score

Although there is no score keeping in this game, the number of rides yet to visit is displayed on the top of the screen. In addition, there is a timer on the top of the screen displaying the number of seconds elapsed.

5. Implementation

In the C implementation of RVO2, the agent is defined as a struct. Additional properties are added for this game project.

5.1 The Modified Agent Struct

In the RVO2 library, the agent is defined as a struct with these properties.

- maxSpeed
- radius
- newVelocity
- position
- preferredVelocity
- velocity

The following additional properties are added for this game.

- whether the agent is staying within the ride
- which ride the agent is currently in
- the ride that the agent is going to next
- how long the agent has been pursuing the current goal
- the time limit of the agent in pursuing a goal

5.2 A Ride Class

A Ride class is created in Objective C for this game because the rides in the game cannot be represented by simple goals, each of which is defined as a 2D point. With the Ride class, each ride object has an entrance location that the agent is going to that ride has to reach first. In additional, each ride has multiple subgoals. Once an agent has reached the entrance of the ride, it will repeatedly select a random subgoal within that ride until the random subgoal is to leave the ride. In this case, the agent will exit the ride through the ride's entrance and go visit another ride. This allows simulation of agents wandering in the ride area.

5.3 Isometric View and Z-order

With isometric view, the z-order of the graphics will then need to be sorted by their ground line's y-coordinates. To do this, the z-order of all the agent and ride sprites are grouped in an array and sorted together by their y-coordinates using `sortUsingFunction` at frame rate. The sprites are drawn on screen in their order in the sorted array.

The y-coordinate of a sprite is based on its center, which is often not the ground line for the sprite. By subclassing Cocos2d's CCSprite, I can define the sprite's registration point (i.e. the ground line). By adding functionalities to keep track of the y-coordinate of the registration point, the z-ordering can then be performed properly.

5.4 Strategies to Create the Wander Effect for My Game Environment

To avoid the crowd pattern shown in Figure 1, the following strategies are used.

- Add goals other than the rides' locations to the list of goals that an agent can select from. These non-ride goals can be:
  - on the walkway
  - in open areas where there are no rides
- Add subgoals within the ride area to hold the agents for a while. Once an agent has reached the entrance of the ride, it will repeatedly select a random subgoal within that ride until the random subgoal is to leave the ride. This forces the agents to wander within a small ride area for a random period of time.

Another strategy is to set a new random goal whenever the agent collides with an obstacle. The demo [5] shows a pretty convincing wandering pattern of an agent. This technique, however, is not used in this project because the RVO2 library has the obstacle avoidance algorithm built-in.

5.5 Ensure Agents' Initial Random Positions to be Outside of Obstacles

When the game starts, agents are placed at random positions. However, their positions have to be outside of any obstacles. Otherwise, the agent will be trapped inside an obstacle. To ensure an agent's random position is outside any obstacles, the random position is checked if it is inside any obstacles. A new random position is generated until the position is outside all obstacles.

The obstacles are polygons. Here, the ray casting algorithm [7], [8], [9] that is used to determine if a point is in the interior of a polygon is used. The basic idea is to draw a horizontal line through the point and determine the number of intersections between the horizontal line and the polygon edges. If the number of intersection points on both sides of the point is odd, then the point lies in the interior of the polygon. If both are even, the point lies outside the polygon.

6. Frame Rate

In the early stage of development, the RVO2 implementation did not have the spatial partitioning for the agents and obstacles. A simulation with 100 agents without any obstacles ran at 30 fps on an iPod Touch 4G, and 60 fps on an iPhone simulator on a MacBook Pro. With the newer version of RVO2, which has implemented Kd-tree for the agents and obstacles, the game runs at 60 fps even with 150 agents if without obstacles.

The game now runs in the range of 45-55 fps with 120 agents and obstacles (total about 40 vertices) with the Kd-tree turned on for both agents and obstacles. The same game with the Kd-tree turned off for the obstacles would run at about half the frame rate.

7. Problems and Problem-Solving

During the course of project development, it has been observed that agents get stuck at the obstacle edges. This side effect, although may not be desirable for other situations, actually works well for the context of this game. In a theme park environment, the guests often stop and gather at certain rest areas. The crowd gathering may cause congestion at narrow passages. However, this happens in real life too. Some obstacles have been repositioned to lessen the crowding.

Another crowding problem is that agents can get trapped in the middle of a crowd. A deadlock has been observed such that the agents at the outer edge of the crowd want to get to the center (where their goals are located) while the agents in the middle want to get out (because they have reached their goals.) To alleviate the problem, I added properties to the agent to keep track of how long it has been trying to reach its goal. If the time exceeds the specified limit, the agent will randomly select another goal. This allows the agents at the outer edge of a crowd to eventually time out of pursuing the goal that is inside the crowd. The agents will then be redirected to another goal and thereby opening up the crowd to release the agents that have been trapped inside.

At the time of the development of this game and the writing of this report, the C implementation of the RVO2 library is still under development. With the current version, some combinations of the vertex locations of the obstacles are not accepted by the Kd tree for the obstacles. Therefore, some of the obstacle locations, such as the lake at the bottom of the iceberg, have to be removed. In some screenshots, you may see agents walking on the lake.

8. Conclusion and Future Work

I have achieved the goal of using RVO2 in an iPhone game. During the process, I have added extra properties and functionalities to the RVO2 library to suit the gameplay purposes. The RVO2 library provides example codes to demonstrate how to call the RVO2 functions to set up and run the simulation. The codes are very helpful but have to be rewritten in Objective C so they can be integrated with the main code of the game. In addition to these technical aspects, I have also implemented many extra gameplay features to make the game more fun to play.

Future work may include:

- Adding spatial sound for each ride. Rides will have audio, for example, the waterfall has the water sound. The amplitude and the balance of the left/right channels depend on the player character's location relative to the ride. This will hep the player locate the player character.
- Let the player enter the number of agents in the game. From the gameplay point of view, this is like providing levels. From the performance testing point of view, it will allow me and the RVO developers to interactively change the number of agents and observe the frame rate.
- Porting the game to 3D. Since RVO2 is for 2D crowd simulation, I started the game project as a 2 game. The game can be ported to 3D using a 3D game engine, in which case the rides will be 3D models and the z-ordering will be taken care of by the 3D game engine although the agents will still be moving in a 2D plane.
- Make use of the gyroscope to control the direction and speed of the player character.
- Make use of the accelerometer: The player can shake the device to create chaos of the crowd.
- Make a larger map with more rides.
- Add the peer-to-peer functionality to allow two players to play together.
- Add ambient props, such as pigeons
- Create a class of rides that has a set start and end time, like a show. It will release the agents within the ride simultaneously. The observer design pattern will be well-suited for this--the ride being the subject and the agents being the observers.
- Add AI elements: Encode the personality traits (such as curious, shy, competitive, impatient) into simulation parameters, such as agent radius, speed, how frequent the agent will change goal, the time limit of an agent in pursuing a goal.

- Use the same game in a different context, for example, puppy training of come and heel in a dog park, where all agents are puppies in the dog park and you have to take control of your puppy to go reach all the goals as quick as possible.

9. References

[1] http://gamma.cs.unc.edu/RVO/ and http://gamma.cs.unc.edu/RVO2/
[2] Stephanie Huerre, "Agent-based crowd simulation tool for theme park environments," *CASA 2010*, Saint-Malo, France, May 31-June2, 2010.
[3] http://www.youtube.com/watch?v=lfyneib5UQE
[4] http://www.cocos2d-iphone.org
[5] Ian Millington, *Artificial Intelligence for Games*. Morgan Kaufmann: 2002.
[6] Allison Drulin, "Designing Online Interactions: What Kids Want and What Designers Know," *ACM Interactions*, vol. XV.3, May + June 2008.
[7] http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html
[8] http://paulbourke.net/geometry/insidepoly/
[9] http://en.wikipedia.org/wiki/Point_in_polygon

10. Addendum

After the project presentation, Professor Manocha suggested having the agents in different colored shirts (instead of all in plain white shirts) and making the player character stand out. Figure 7 shows the map with the changes and Figure 8 shows the tentative title screen for the game. The player character is now a grown-up. All the other agents look like kindergarten kids in comparison to the player character. With the player character being so different from the rest of the crowd, I feel I should provide a back-story for the game to explain this. So the back-story is this:

"You (the player character) are a kindergarten teacher taking your kids on a field trip to a theme park. You have arrived too early. All the rides are not open yet. Your kids are disappointed. You now need to go to all the rides to start them one by one. (And yes, somehow a kindergarten teacher is allowed to go in the rides to start them for his/her kids! What a theme park!)"

I think this back-story matches the gameplay well because after the player character has reached a ride, an animation of that ride will start. It really looks like the player character starts the ride. Now to match the game elements with this new back-story, I also change the sound effect from "ta-da" to kids yelling "yay!" when the player character has reached a ride. In addition, the alert prompt has now changed from "Yeah! You have been to the [ride name]!" to "Yeah! You have opened the [ride name]!".

(a)

(b)

Figure 7. New changes are made after the project presentation. The agents' shirts are in 15 different colors and the player character (circled in red in (a)) stands out of all the other agents. (a) Full map; (b) A screenshot of the game.

Figure 8. Tentative title screen for the game.