Modeling, Designing, and Implementing an Avatar-based Interactive Map

Stefan Andrei
Lamar University, Beaumont, TX 77710, U.S.A.
Stefan.Andrei@lamar.edu

Milin Joshi
California State University, Long Beach, CA 90840, U.S.A.
milincjoshi@gmail.com

ChandrakantRudani
Lamar University, Beaumont, TX 77710, U.S.A.
Rudani.c@gmail.com

Ankur Shah
Lamar University, Beaumont, TX 77710, U.S.A.
Ashah8@lamar.edu

BharatkumarTejwani
Lamar University, Beaumont, TX 77710, U.S.A.
btejwani@lamar.edu

Abstract

Designing interactive maps has always been a challenge due to the geographical complexity of the earth's landscape and the difficulty of resolving details to a high resolution. In the past decade or so, one of the most impressive map-based software application, the Global Positioning System (GPS), has probably the highest level of interaction with the user. This article describes an innovative technique for designing an avatar-based virtual interactive map for the Lamar University Campus, which will entail the buildings' exterior as well as their interiors. Many universities provide 2D or 3D maps and even interactive maps. However, these maps do not provide a complete interaction with the user. To the best of our knowledge, this project is the first avatar-based interaction game that allows 100% interaction with the user. This work provides tremendous help to the freshman students and visitors of Lamar University. As an important marketing tool, the main objective is to get better visibility of the campus worldwide and to increase the number of students attending Lamar University.

Keywords: interactive map, avatar-based game, marketing tool, software, virtual reality.

1. Introduction

Most institutions providing so-called 'interactive maps' only have zoom-in capability and left-right and up-down cursors. These maps do not provide digitized photos from real pictures nor buildings' interiors. Among these interactive maps, the Global Positioning Systems (GPS) have perhaps the highest interaction in the sense of communicating with satellites to detect the approximate position of the GPS' user. However, these systems provide only navigation in the exterior of the buildings, and they have a certain approximation. It is known that it is very challenging to model and design a 100% interactive map. This article describes an innovative technique to model and design exteriors, interiors, and the distances between buildings. The user will be modeled by an avatar carrying a hover-board on his/her back. The exterior and interior of the buildings are photographed and then digitized to create the real feeling of the building.

It is believed that this work is of tremendous help to the freshman students, potential students, and visitors of Lamar University. It is very likely that the freshman students are not familiar with the buildings' locations and general departments of interests, such as Health Center, Records Office, Admission Office, Dining Hall, Library, and many more. Despite the fact that

interactive maps are already planted across the campus, it takes time to get the exact path to the destination. Hence, the most important advantage of using this interactive navigation tool is saving time for the students and making them independent. With the use of this avatar-based game, new visitors/guests/residents can take virtual tours, which will be available through the Lamar University Campus' website. By utilizing the innovation, he/she can get to know the Lamar University Campus at any moment of time from anywhere in the world. We think that this project is an important marketing tool because it will have a positive impact due to at least two major objectives:

1. Allows better visibility of Lamar University worldwide;
2. Increases the number of students attending Lamar University.

2. Background

This project is built using the `Unity5` software package, one of the best platforms offered for developing 2D and 3D games due to its interactive experiences and high-end content for users[1]. Developing this project has its own benefits, such as its deployment over a number of platforms available in the current market of Smart-Devices, such as iOS, Android, Windows Phone, Windows PC, MAC OS, LINUX/UNIX Web Browsers, Samsung Smart TV, Android TV, XBOX One, XBOX 360, PS4, PS3, PS Vista, Wii U, Blackberry, Windows Store, and Virtual Reality platform, like Oculus Rift Gear.

`Unity5` has its competitors in the current market, such as UnReal Engine, GameMaker Studio, and CryEngine. Choosing `Unity5` for this project was the most suitable choice because it offers three programming languages: C#, JavaScript, and Boo. It also supports assets from major 3D modeling software applications like Autodesk Maya, Cinema4D, 3Ds Max, Softimage and Blender.

The objectives of this innovation are to accurately simulate a tour, model the dynamic look, and feeling of each and every asset around the Lamar University Campus. The next section presents the methodology and the data structures implemented, while developing the avatar-based game.

3. The methodology and the data structures

This project uses `Unity5`'s Navigation and Path finding systems to reach the selected destination along with several different components, which makes `Unity5` software unique compared to different 3D game software. There are certain definitions from `Unity5` software, which we would like to mention in order to help understanding the implementation of our project. The avatar-based navigation system allowed us to create characters, which can navigate the game world using the `Unity5 NavMesh` system. Our project uses the `Unity5` navigation system to give the game character (or agent) the ability to understand that they need to take the stairs for reaching the second floor of a building.

The `NavMesh` (the data structure for Navigation Mesh) is an *abstract* data structure largely used in artificial intelligence for representing the walkable surfaces of the game world and finding paths from one walkable location to another. It is abstract because it can be later implemented for many particular navigation meshes. The data structure is built (also called baked) automatically from your level geometry.

A *navigation mesh* is a collection of two-dimensional convex polygons that define the portion of an environment traversable by agents. Convex polygons represent a useful representation, since we know that there are no obstructions between any two points inside a polygon. In addition to the polygon's boundaries, we store information about which polygons are neighbors to each other. This allows us to examine the whole walkable area. Path finding between polygons in the mesh can be done with one of the large number of graph search algorithms, such as **A***. In robotics this mechanism of linking two-dimensional convex polygons is known as *meadow mapping*. This concept is actually already implemented in `Unity5`. To find the path between two locations in a scene, we first need to map the start and destination locations to their nearest polygon(s). The software may initialize searching from the start location, visiting all the neighbors until it reaches the destination polygon. Tracing the visited polygons allows the software to find the sequence of polygons, which will lead from the start to the destination.

[1] Unity Community- Unity Technologies. www.Unity3d.com/commUnity, [accessed on February 20, 2015]

The `NavMesh` agent component helped us make characters, which avoid each other while moving towards their goal. Agents reason about the game world using the `NavMesh`, and they know how to avoid each other as well as moving obstacles.

The `Off-MeshLink` component helped us to incorporate navigation shortcuts, which cannot be represented using a walkable surface. For example, opening a door before walking through it can be described as an off-mesh link.

The `NavMeshObstacle` component describes moving obstacles the agents (also known as characters) should avoid, while navigating the world game. A door or building wall controlled by the physics system is a good example of an obstacle. While the obstacle is moving, the agents do their best to avoid it. However, once the obstacle becomes stationary, it will carve a hole in the `NavMesh`. Hence, the agents can change their paths to move around it. Alternatively, if the stationary obstacle is blocking the path way, the agents can find a different route.

When we want to design an intelligent software able to move characters in the game, at least two problems have to be solved:

1. how to *compute* the level (i.e., the floor number or altitude) to find the destination
2. how to *move* there.

Despite the fact that these two problems are tightly coupled, they are quite different in nature. The problem of determining the level is more global and static due to the fact that it takes into account the whole scene. Moving to destination is more local and dynamic because it only considers the direction to move.

The navigation system needs its own data to represent the walkable areas in a game scene. The walkable areas define the places in the scene where the agent can stand and move. In `Unity5` the agents are described as cylinders. The walkable area is built automatically from the geometry in the scene by testing the locations where the agent can stand. The locations are connected to a surface laying on top of the scene geometry. The sequence of polygons which describes the path from the start to the destination is called a *corridor*. The agent will reach the destination by always heading towards the next visible corner of the corridor. In addition, the agent will find all the corners of the corridor in one swoop and animate the character to move along the line segments connecting the corners. The steering logic takes the position of the next corner in order to figure out a desired direction and speed (or velocity) needed to reach the destination.

Obstacle avoidance chooses a new velocity, which balances between moving in the desired direction and preventing future collisions with other agents and edges of the navigation mesh. `Unity5` is using *Reciprocal Velocity Obstacles* (RVO) to predict and prevent collisions. Finally, after steering and obstacle avoidance, the final velocity is calculated. In `Unity5` the agents are simulated using a simple dynamic model, which also takes into account acceleration to allow more natural and smooth movement.

At this stage it is possible to alter the velocity from the simulated agent to the `Mecanim` animation system to move the character or let the navigation system take effect. Once the agent has been moved using either method, the simulated agent location is moved and constrained to `NavMesh`.

One of the most important things to understand about having a robust navigation is the difference between *global* and *local navigation*. Global navigation is used to find the corridor across the world. Finding a path across the world is a costly operation requiring quite a lot of processing power and memory. The linear list of polygons describing the path is a flexible data structure for steering, and it can be locally adjusted as the agent's position moves. In contrast, local navigation tries to figure out how to efficiently move towards the next corner without colliding with other agents or moving objects. In a nutshell, the cost allows the user to control the areas favored by the pathfinder when looking for a path. For example, if you set the cost of an area to, say, 3.0, traveling across that area is considered to be three times longer than alternative routes. To fully understand how the cost works, information in regards to the pathfinder must be provided.

`Unity5` uses A* algorithm to calculate the shortest path on the `NavMesh`. A* works on a graph of connected nodes. The algorithm starts from the nearest node to the path start and visits the connecting nodes until the destination is reached. Since the `Unity5` navigation representation is a mesh of polygons, the first thing the pathfinder needs to do is to place a point on each polygon, which is the location of the node. The shortest path is then calculated between these nodes. The cost

to move between two nodes depends on the distance of travel and the cost associated with the area type of the polygon under the link, which is *distance* * *cost*. For example, if the cost of an area is, say, 2.0, the distance across the polygon will appear to be twice as long. The A* algorithm requires that all costs must be larger than 1.0. The effect of the costs on the resulting path can be hard to tune, especially for longer paths. The best way to approach costs is to treat them as hints. For instance, if someone wants the agents to not use `Off-Mesh` links too often, then their cost should increase. However, it can be challenging to tune a behavior where the agents prefer to walk on sidewalks.

A point is *intermediary* if it is an intersection of two non-collinear lines. Denoting the number of intermediary points from the source point to the destination point by n, the A* algorithm[2] has a worst time complexity of $O(n^3)$. [3]

Another important concept is that the pathfinder does not always choose the shortest path. The reason for this is the node placement. The effect can be noticeable in scenarios where big open areas are next to tiny obstacles, which results in navigation mesh with very big and small polygons. In such cases the nodes on the big polygons may get placed anywhere in the big polygon, and from the pathfinder's point of view, it looks like a detour. The *cost* per *area type* can be set globally in the *Areas* tab.

The following main data structures from the `Unity5` project have been used in our project:

1. `ArrayTools` - functions to use built-in arrays like a Java `ArrayList;`
2. `DeepCopy` - creates a deep copy of an array or hashtable;
3. `MultiKeyDictionary` – similar to a `Dictionary`, but with two keys for each element;
4. `ObjectCopier` - an alternative to `IClonable;`
5. `ParallelKeyDictionary` - like a `Dictionary`, but you can have identical keys;
6. `ReflectedObject` - Reflects a target object, providing quick access by name to reading/writing its fields/properties, and calling its methods;
7. `Set` - A set data structure.

Let us briefly describe some considerations about converting an analog photo into a digital photo. The main idea is to create a gray scale image exported from MicroDem in Gimp[4].The next step is to create a new 3Dterrainobject in `Unity`, which supports 3D Texture[5] use and creation from the shader and the script. While use cases of 3D Textures might not seem as straightforward at first, they can be an integral part of implementing specific kinds of effects such as 3D Color Correction. The following snippet of Java-like pseudo-code creates a "neutral" 3D texture[6]:

```
varnewC : Color[] = new Color[dim * dim * dim];
varoneOverDim : float = 1.0f / (1.0f * dim - 1.0f);
for(vari : int = 0; i< dim; i++)
for(var j : int = 0; j < dim; j++)
for(var k : int = 0; k < dim; k++)
newC[i + (j*dim) + (k*dim*dim)] = new
Color((i*1.0f)*oneOverDim,
  (j*1.0f)*oneOverDim,
  (k*1.0f)*oneOverDim, 1.0f);
```

[2] ***, The A* Pathfinding Project for Unity 3D. http://arongranberg.com/astar/docs/graph-updates.php, [accessed on February 29, 2016]; Zeng & Church, 2009.
[3] Posts and Topics - Unity Technologies.http://answers.unity3d.com/questions/767153/a-pathfinding-questions-1.html, [accessed on February 29, 2016]
[4] Importing GIS in Unity. http://electricarchaeology.ca/2015/06/08/importing-gis-data-into-unity/, June 8, 2015 [accessed on February 29, 2016]
[5] 3D Textures - Unity Technologies. http://docs.unity3d.com/Manual/class-Texture3D.html, [accessed on February 29, 2016]
[6] Historical Maps into Unity3D. http://electricarchaeology.ca/2015/06/09/historical-maps-into-unity3d/, June 9, 2015 [accessed on February 29, 2016]; Li & Wang, 2013

4. The Agile software development process and its UML diagrams

The software process on which this project was developed is the well-known agile development model. The agile model has five phases: designing, building, configuring, testing, and releasing prototype versions. Compared to other software development processes, the agile process follows a continuous improvement cycle, exposing flaws earlier and reducing the development time. As a consequence, its value is achieved faster as software versions arrive to the customer more frequently. In conclusion, the advantages of using the agile development model are shorter development cycles, early departmental feedback, and its continuous improvement. Here is a list of guidelines that we used in our project's implementation and analysis:

- Any difficulty in design, coding and testing a modification should signal the need for redesign or re-coding.
- Modifications should fit easily into isolated and easy-to-find modules. If they do not, some redesign is possibly needed.
- Modifications to tables should be especially easy to make. If any table modification is not quickly and easily done, redesign is indicated.
- Modifications should become easier to make as the iterations progress. If they are not, there is a basic problem, such as a design flaw or a proliferation of patches.
- Patches should normally be allowed to exist for only one or two iterations. Patches may be necessary to avoid redesigning during an implementation phase.
- The existing implementation should be analyzed frequently to determine how well it measures up to project goals.
- Program analysis facilities should be used whenever available to aid in the analysis of partial implementations.
- User reaction should be solicited and analyzed for indications of deficiencies in the current implementation.

These are true advantages of the agile model:

- The feedback from early versions improves later stages.
- The possibility of changes in requirements is reduced because of the shorter time span between the design of a component and its delivery.
- Users get benefits earlier than with the conventional approach.
- Early delivery of some useful components improves software cost because a portion of return investment can be received early.
- Smaller sub-projects are easier to control and manage.
- "Gold-plating", which are requesting features that are unnecessary and not in fact used, is reduced because users will know that if a feature is not in the current increment it can be included in the next version.

The next section describes the Unified Modelling Language (UML) diagrams designed for the project, which are a state diagrams (also known as statecharts) for the `Player` movement, the `Navigation` system (Figure 1). In addition, we used a class diagram for the `Player` and `Camera` movement (Figure 2). When the avatar-based game starts, the state of the `Player` is `Idle`, i.e., `Player_IDLE`. When the user selects the building, it enables the navigation path towards the destination. If the user selects any arrow keys (Right, Left & Up) the state of the player will change to running (i.e., `Player_Running`). Also, the path will diminish along with the player movement; hence, the state of navigation path will change to `Changing_Path`. Furthermore, the path will also be diminishing along with the player's movement, so the state of the navigation's path will change

to `Changing_Path`. If no arrow keys are selected the state of path will again change to idle as the path is not changing now. Thus, if no arrow keys are selected, the state of the player will change to `Player_IDLE`. If the rightmost close button of the game is selected, the avatar-based game will stop.

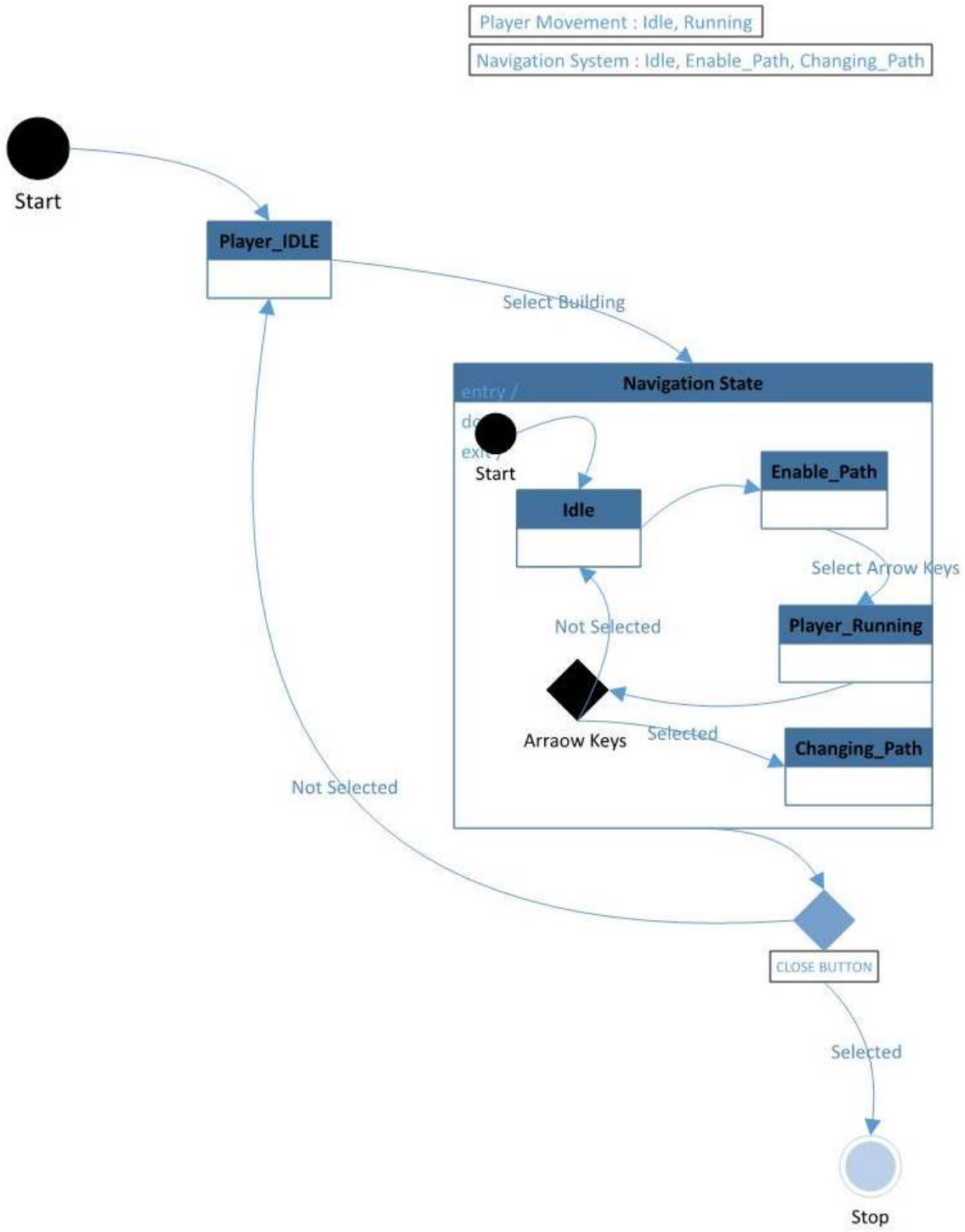

Figure 1. The Statechart for the `Player` movement and the `Navigation` System

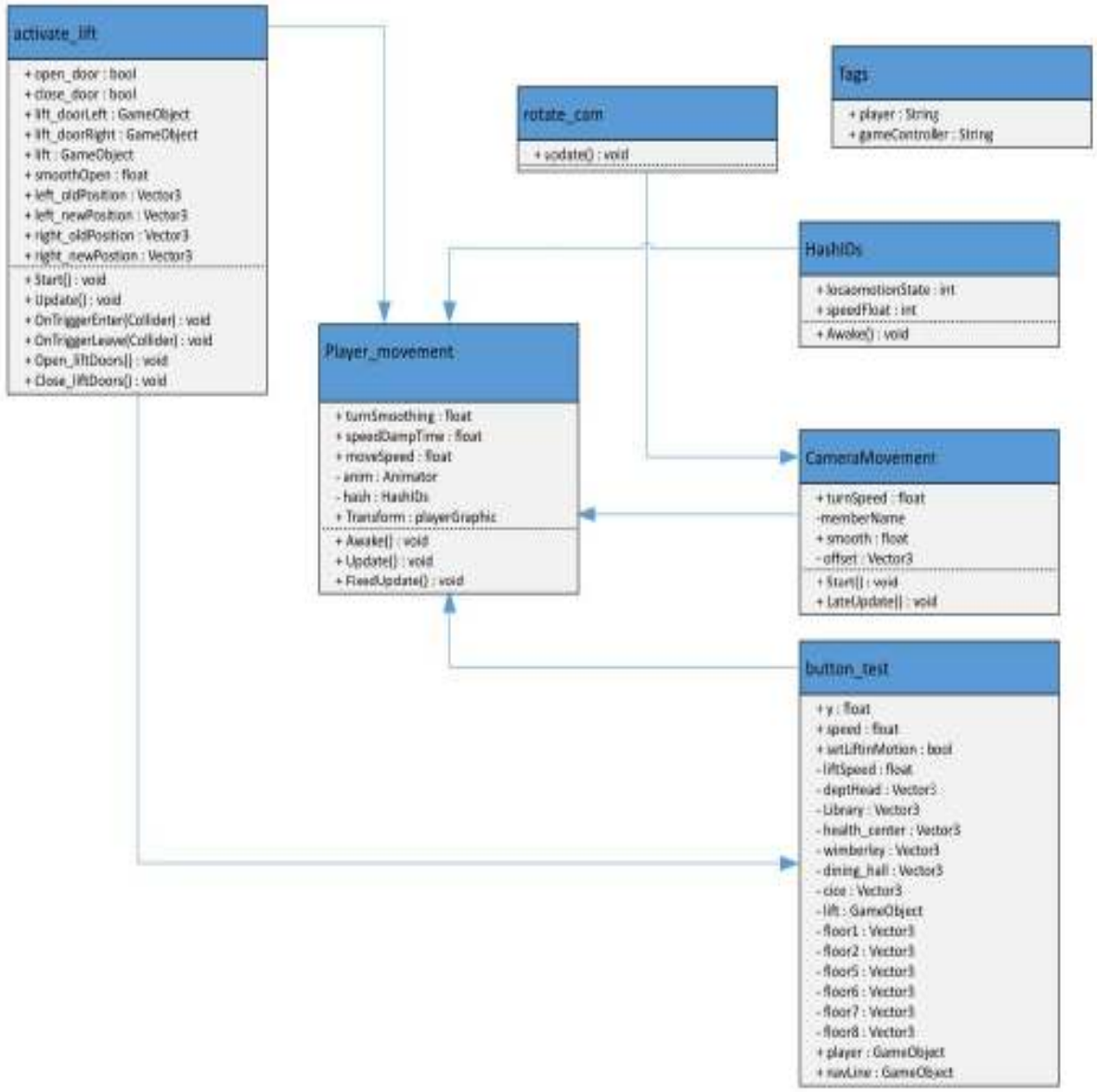

Figure 2. The class diagram for the `Player` *and* `Camera` *movements*

5. Experimental results

This section is dedicated to show the screenshots of our implementation for this project. In particular, we incorporated the following buildings from the Lamar University Campus:

1. The Maes Building
2. The Students Health Center
3. The Records Office
4. The Wimberley Building (The Admission Office, The International Student Services Office, etc.)
5. The Dining Hall
6. The Mary and John Gray Library
7. The Center for Innovation, Commercialization, and Entrepreneurship (CICE) Building

Figure 3 shows an avatar ready to start the game. The user has the option to click the button called 'Go Cardinals! Start' Button. Once that happens, the avatar gets to choose going to one of the buildings of interest mentioned above.

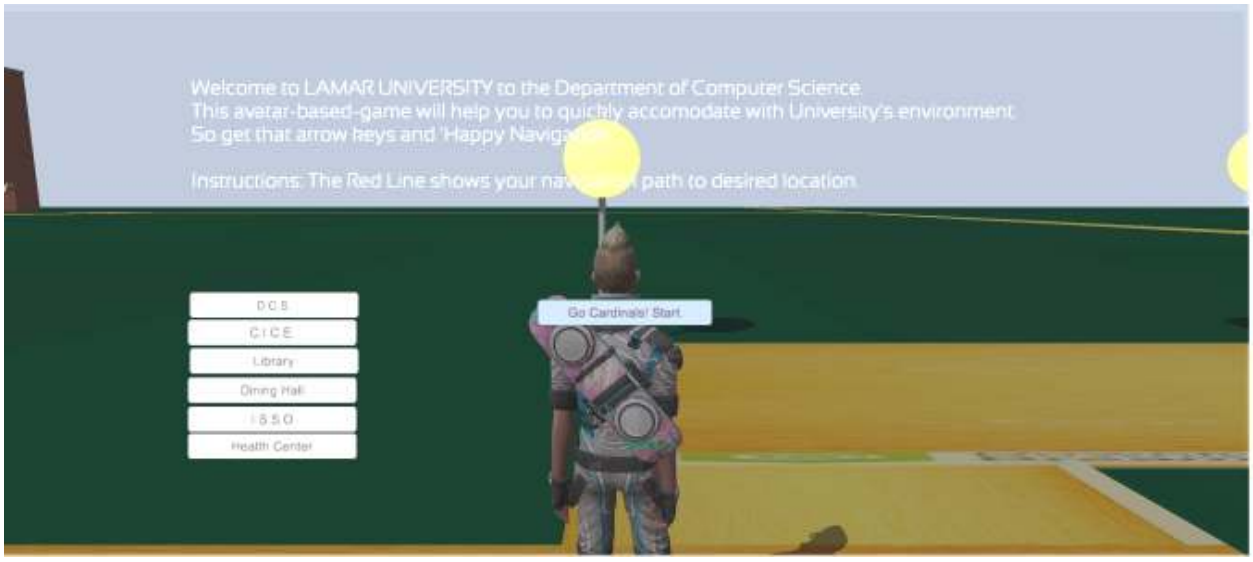

Figure 3. The 'Welcome Screen' of our implementation

Figure 4 represents the navigation path to the Department of Computer Science inside the Maes building after selecting the option "D.C.S.", which stands for Department of Computer Science.

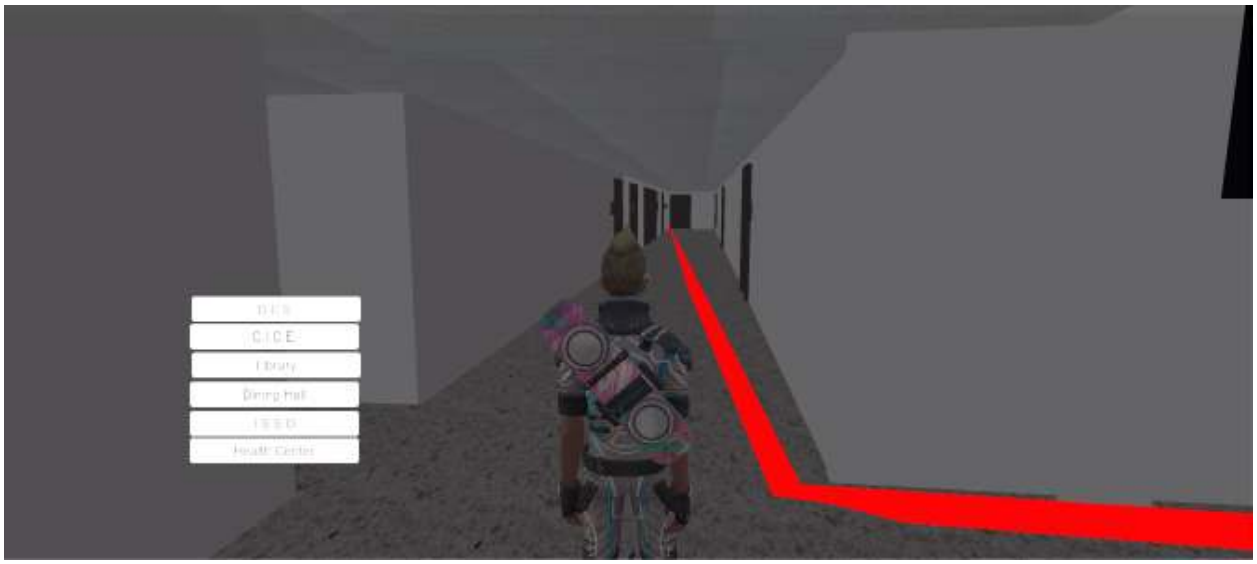

Figure 4. An inside view of the "Maes building along with the Navigation Path"

Figure 5 represents the avatar standing outside and in front of the Mary and John Gray Library after selecting the option "Library". The library's main purpose is to facilitate students with a variety of scholarly information within the overall composition of the University's stated mission. Figure 5 shows the path generated by A* algorithms with a red color.

Figure 5. A front view of the "Mary and John Gray Library"

Figure 6 exhibits the ambience of the study environment that allows students to have group discussions, and when to access Internet, and more. The photographs have been digitized in a very realistic way.

Figure 6. An inside view of Mary and John Gray Library

6. Conclusion and future work

This article introduced the modelling, designing, and implementation of the avatar–based virtual campus tour. It has thoroughly explained the design, methodology, implementation, and working set of the game. In our regular periodic meetings, we followed an agile software developing process by showing early releases to colleagues interested to invest in this ambitious project. With rapid prototyping, we converted each physical building into avatar-based virtual buildings. Our team is dedicated to working on future enhancements and constantly keeping an eye on new requirements that can be helpful to everyone.

The following improvements may be considered as a future work: an autonomous navigation, an Android application, designing all buildings of the Lamar University Campus, and public website access (on the Lamar University's public domain to be available online).

We will introduce a novel path planning and replanning algorithm the two-way D∗ (TWD∗) algorithm based on a two-dimensional occupancy grid map of the environment based on a two-dimensional occupancy grid map of the environment.

References

Wadhwa, B., Andrei, S., & Jien, S.Y. (2007). Software Engineering. An Object-Oriented Approach. McGraw-Hill, Revised Edition.

Unity Community- Unity Technologies. www.Unity3d.com/commUnity, [accessed on February 20, 2015]

Categories of Services - Unity Technologies. blogs.Unity3d.com/category/services/, [accessed on March 12, 2015]

Navigation Mesh Agent - Unity Technologies. docs.Unity3d.com/Manual/class-NavMeshAgent.html, [accessed on April 5, 2015]

***, The A* Pathfinding Project for Unity 3D. http://arongranberg.com/astar/docs/graph-updates.php, [accessed on February 29, 2016]

3D Textures - Unity Technologies. http://docs.unity3d.com/Manual/class-Texture3D.html, [accessed on February 29, 2016]

Importing GIS in Unity. http://electricarchaeology.ca/2015/06/08/importing-gis-data-into-unity/, June 8, 2015 [accessed on February 29, 2016]

Historical Maps into Unity3D. http://electricarchaeology.ca/2015/06/09/historical-maps-into-unity3d/, June 9, 2015 [accessed on February 29, 2016]

Posts and Topics - Unity Technologies.*http://answers.unity3d.com/questions/767153/a-pathfinding-questions-1.html*, [accessed on February 29, 2016]

Li, S. & Wang, H. (2013). Realistic 3D Face Modeling and Application Research. International Journal of Electronic Commerce Studies, Vol. 4, No. 2, pp. 203-212.

Zeng, W. & Church, R.L. (2009). Finding shortest paths on real road networks: the case for A*. International Journal of Geographical Information Science,Vol. 23, No. 4, pp. 531–543.