Drunk Stride

(Artificial Intelligence for Video Games project)

Professors:

- ***Dario Maggiorini***
- ***Davide Gadia***

Made By:

Leandro Bognanni

student ID: (34935A)

Date: 18/01/2024

UNIVERSITÀ DEGLI STUDI DI MILANO

Table of Contents

1 Overview

1.1 Project Objective

The project's focus is to create an agent that roams on a platform without ever moving in a straight line.
This project has been developed in Unity, using the C# language to create the scripts.
The project also aims to evaluate, during the exam, the knowledge acquired in the 'Artificial Intelligence for Video Games' course.

1.2 Setup & Constraints *(from project assignment)*

The platform can be of any size; square or rectangular.
The agent will change trajectory at random intervals. Each time the agent will travel over a circumference leading first to right then to the left, then to the right again … and so on.
The agent is moving at a constant speed of 1 meter per second. At each interval, the agent will pick a random value for the time to the next trajectory change in the range (0, 10] seconds. Note that 0 is excluded. Then, the agent selects a random circumference leading right or left with a radius between 0 (excluded) and the maximum radius that is not making the agent fall off the platform.
The selection of the radius is independent from the time of the next trajectory change.
The agent never stops moving.
The system must work independently on the platform shape or size.

1.3 Specification Analysis

1.3.1 Agent's Knowledge

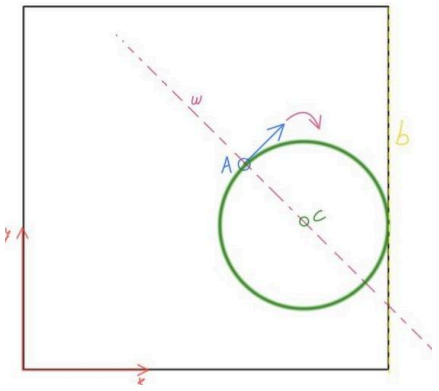
The motion trajectory of the agent, and consequently its ability to avoid falling from the platform, must be independent of the shape and size of the platform itself. Therefore, the agent needs information about the platform it is on, in order to move within its limits.

While it is true that the time it has to walk may seem to be part of the agent's internal knowledge, on the other hand, it is also true that this should not be taken into consideration by the AI to calculate its trajectory. This effectively designates the ‘time until the next trajectory change’ as belonging to the NPC, but not usable as internal knowledge to take a decision on the path to walk. Instead, it can only be used to understand when it will be necessary to calculate the next 'path'.

1.3.2 Maximum Radius

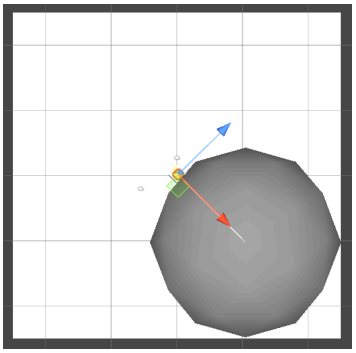
The project's assignment states that the trajectories to follow are circular and chosen with a radius ranging from 0 to 'the maximum radius that is not making the agent fall off the platform', where the radius equal to zero is excluded (further details in paragraph 2.4).
Since there is no reliance on knowledge of the time it will take to traverse the chosen circumference, the maximum radius must be sought without this information. To achieve this, it has been decided to geometrically calculate the largest possible radius that allows 'drawing' a circumference that, even if traversed in its entirety, will not cause the agent to fall off.
The mathematical calculations performed to determine the center of the circle with maximum radius can be described by the following system of equations:

$$\begin{cases} dist(A,\ C)\ =\ dist(C,\ b) \\ C\ \in\ w \end{cases}$$

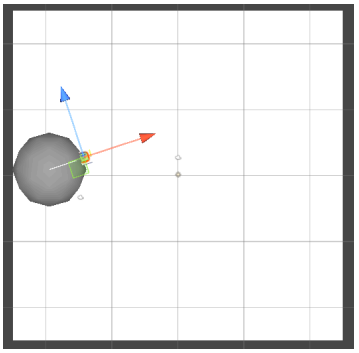


where, *A* represents the position of the agent on the platform, *C* is the point on the plane where the center of the circle is located, *b* indicates the border of the platform for which the calculations are being carried out, and *w* is the line representing the direction in which the agent must walk (the equation of the line perpendicular to where the NPC is looking).
In other words, the system seeks the coordinates of *C* in such a way that the circle has its center in the direction *w*, and the distance between the center and the agent is equal to the distance from the center to the reference edge.

These calculations use the plane as a reference, more precisely, the Cartesian axes are positioned in the south-west corner of the platform. This generic formula will then be implemented in the code in different variations that take into account specific cases (such as the slope of *w* being equal to 0 or infinite) and considering that the edges are parallel to the axes.
These calculations lead to the discovery of 2 possible values for the coordinates of the center. To choose the correct result, it is necessary to take into account the agent's direction (right/left). Once the calculation is performed for each of the 4 edges of the platform, it is sufficient to choose the smallest circle among those found, to obtain the maximum radius circle discussed earlier.

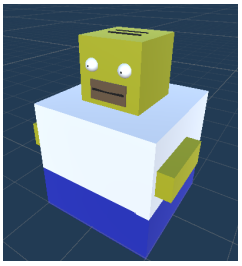


maximum radius examples



2 Project Design

In this section, the various implementation choices will be detailed.
Note: The GameObject 'CuberSimpson,' the protagonist of the project, will often be referred to as 'the agent' or 'the NPC'.



2.1 Key Design Elements

The central concept behind the implementation of the agent's behavior draws inspiration from movement algorithms that use a second in-game element as a 'destination.' In the implementation of the assignment, a

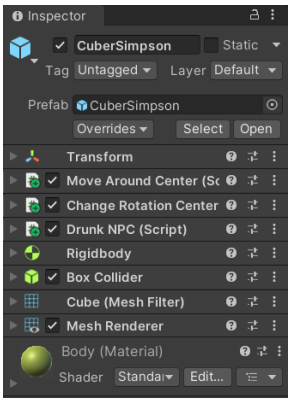
GameObject named 'RotationCenter' was thus employed to be the center of the agent's circular motion.
The RotationCenter is a small and flat cylinder without any collider and no particular component attached, it can also be disabled to make it invisible and act just a support element.
The implementation of the project has been divided into three main scripts that collaborate to achieve the desired behavior, along with a script used to create a data structure, and a fifth one used for camera management.
The main scripts have three distinct roles: representing the agent's internal knowledge and providing access to it, moving the agent around the center with a uniform circular motion, changing the position of the RotationCenter based on the direction whenever the random time interval elapses.
Another key element of the design was the introduction of a "Pivot" 'Empty'-GameObject as a child of the platform; this element represents the origin of the axes of the reference system used for calculations on the maximum radius.
Note: In the project implementation, it was decided to declare class attributes as 'private' to preserve the object-oriented programming paradigm. Therefore, the '[SerializeField]' tag was added to expose them to the Unity inspector when necessary.

2.2 Components

In this section, the components belonging to each GameObject in the scene will be displayed, with a particular emphasis on the agent.

2.2.1 The Agent

The agent is described by a prefab made combining different 3D Object 'Cube' and 'Sphere'
The agent has attached the three main scripts: DrunkNPC, MoveAroundCenter, and ChangeRotationCenter.
In addition to these, it has been equipped with a Rigidbody, both to leverage it in the movement script and to obtain the characteristic of being affected by gravity and fall if it goes off the platform "for free".



2.2.2 Others

The other GameObjects present do not have any specific scripts attached, except for the TopDownCamera (see TopDownView script). The platform is represented by a prefab made by combining a 3D Object 'Plane' and an ‘Empty element’, while the RotationCenter is a 3D Object 'Cylinder' whose collider component has been removed.

2.3 Scripts Description

In this section, the functionalities of individual scripts will be detailed.
Note: additional descriptions aiding the comprehension of the code can be found within the code itself in the form of comments, helping to explain the roles of each implemented method and attributes.

- DrunkNPC

This script contains the class describing the internal knowledge and state of the agent.
The main attributes describing the NPC's state are 'rndTimeToWalk' and 'walkDirection', representing the time interval to walk and the direction in which it is walking, respectively. The class then includes some methods useful for obtaining information about the agent from the outside and for controlled state modification. In the 'Start' method inherited from MonoBehaviour, the direction is set to +1 to indicate clockwise movement, and the initial random time interval is generated.

- MoveAroundCenter

This module contains the code that allows the agent to move with a constant tangential velocity around a GameObject (RotationCenter in our case).
At Start, the agent's Rigidbody and DrunkNPC components are obtained, which will be used for implementing the movement in the FixedUpdate method. In the latter, the position and rotation of the Rigidbody are modified (using the MovePosition and MoveRotation methods) based on the rotation radius, the position of the center and the agent’s direction to create the rotational movement.

- ChangeRotationCenter

In this script, all the necessary calculations to modify the position of the rotation center at each time interval take place. To fulfill its task, this script

requires the Transform component of both the RotationCenter and the Platform, along with its Pivot.
The class is primarily composed of private methods used to break down the problem of calculating the circumference with maximum radius and choosing the random center into simpler parts.
In the Start method, the DrunkNPC component is obtained, the time elapsed since the last trajectory change is set to 0, and the position of the center is calculated for the first time and assigned.
In the Update method, the 'elapsed' attribute is incremented by ‘Time.DeltaTime’ to track the time. If the time elapsed is equal to the random time interval, the agent's movement direction is changed, a new position for the center is calculated, and the 'rndTimeToWalk' attribute of DrunkNPC is updated with a new random time. The 'elapsed' attribute is then reset to 0.
The main method of the class, which calculates the change in the center's position with the help of utility methods, is named 'CalcNewCenterPosition()'. This method performs the calculations described in section 1.3.2 for each edge of the platform. It then finds the circumference of the maximum radius (Cmax) by choosing the one with the smaller radius among those calculated. Finally, it generates a random circle with a radius between 0 and Cmax and returns a Vector3 representing the new position of the center in the 3D space.

- GazeDirection

This is a utility class that does not inherit from MonoBehaviour and aims to represent information regarding the direction in which the agent is facing.
This information is determined based on the rotation angle (in degrees) of the agent on the y-axis.
This class is useful for more easily managing the problem of determining which circumference should be calculated for the next time interval. In fact, the direction in which the agent must walk (left/counterclockwise or right/clockwise) alone is not sufficient to determine which circumference it should traverse: the direction in which the NPC is facing is also a fundamental factor.

- TopDownView

This very simple script is responsible for managing the top-down camera in the scene.

At Start, the camera is moved to the center of the platform, rotated to look downward, and translated upward based on the size of the platform to make it always visible.
Note: Since a platform with a size greater than 99 (along either x or z axes) would be invisible from the camera, this script also ensures to keep the size of the platform within this limit.

2.4 Corner Cases

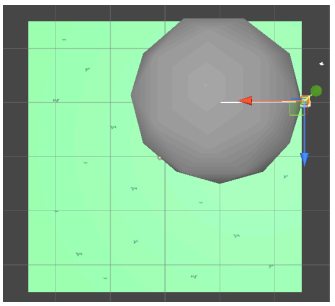
2.4.1 The Edge Problem

The project assignment inherently brings about a problem: what happens if the agent, after walking the entire time interval on a circumference of maximum radius, ends up right on the edge of the platform?
It is a fact that if the agent were to end up exactly on the edge during its wandering at the end of a time interval, it would find itself on the edge and perfectly parallel to it. However, it would be in the situation of having to move towards the void (this happens because reaching the edge involves traversing a circumference tangent to it and then changing direction).
The project specification is clear in stating that the direction of movement must always alternate, and choosing circumferences with a radius of 0 is not allowed. Therefore, the edge problem remains unresolved.
In conclusion, it is not possible to avoid the agent getting stuck or starting to rotate in place once it reaches this particular situation without 'breaking some rules'.
My solution to the problem involves maintaining the same trajectory when the next center to be calculated ends up exactly on the edge.
This way, the agent continues to walk for a sufficient time to calculate the next rotation center in a 'safe' position. Additionally, the speed has been clamped to prevent the agent from rotating too quickly around extremely small circumferences. I believe that among the possible *escamotage* to address the edge problem, I have chosen one of the most reasonable, given the constraints imposed by the specifications; this is because an error in changing direction precisely at the end of the random time interval is already intrinsic in the Update method, which is executed every 'DeltaTime'.

2.4.2 False Start

As evident from the problem explained in the previous paragraph, starting the scene with the agent on an edge can lead to similar outcomes.
It is worth noting, however, that although this position cannot be geometrically reached during a normal trajectory, starting from an edge perfectly parallel to it and with the 'right side free' (having the direction of the NPC perfectly perpendicular to the edge and pointing towards the center of the platform) is allowed.

Here is how the calculation of the maximum radius looks with a 'valid false start':



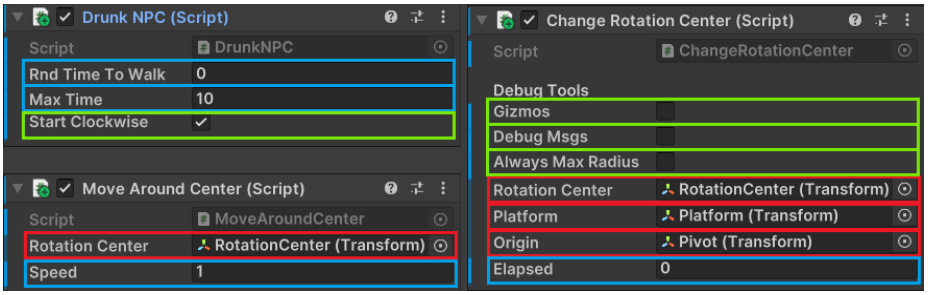
If the agent were to be positioned on an edge with a different rotation than the one described, the clamped speed would come into play, reducing the rotation speed. Eventually, after a bit of waiting, the agent would 'free itself' from the edge by leveraging small calculation errors that occur in determining its position.
Note: Regarding the angles of the platform, it is geometrically impossible to arrive exactly on one of them given the behavior of the agent.
If the agent were to start from a corner it will remain stuck.

2.5 The Scene & Settings

The elements that compose the scene are those described in section 2.2; in addition to them, there is a second camera 'child' of the agent to provide a third-person view.

2.5.1 Fields in the Inspector

The scripts attached to the agent expose various fields in the Unity Inspector.
This is primarily done for three main reasons: linking GameObjects to script attributes (such as providing the RotationCenter to the movement script), enabling the visualization of attributes like the agent's speed or the elapsed time from the inspector (and potentially allowing modification, although this goes beyond project specifications), and making additional functionalities available for debugging purposes and a better visualization of the described behavior.

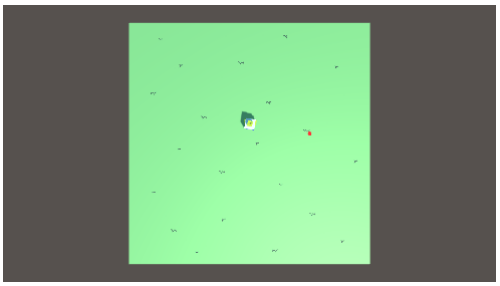


In the figure, the various attributes of the 3 main scripts exposed in the inspector are shown: in red we have those used for linking purposes, in blue those for visualization and in green those useful for debugging.
In particular, the attributes marked in green have the following functionalities respectively: changing the starting direction of the agent (starting on the left), drawing Gizmos in the 'Scene' tab (a sphere denoting the agent's trajectory and the radius connecting it to the center), displaying some debug messages in the 'Console' window (including the calculation and choice of the circumference), forcing the agent to always choose the maximum radius circumference rather than a random one.

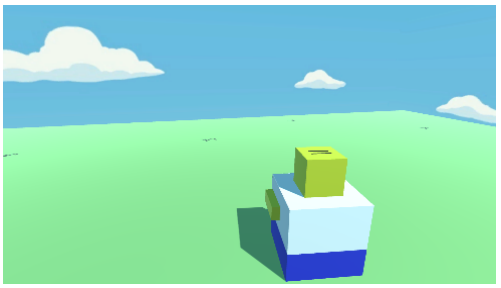
2.5.2 'Play' the Scene

To properly start the scene, first of all, we need to open the "Scenes" folder from the "Project" tab and double click on the "DrunkStride" scene, then it is necessary that all attributes are set correctly (refer to the image in paragraph 2.5.1 for proper initialization).

Additionally, it is subsequently required to position the GameObject ‘CuberSimpson’ above the platform (its y must be equal to that of the platform +1). The RotationCenter can be placed wherever desired (the correct position will be calculated at startup) and can be easily disabled/enabled with the appropriate checkbox to make it invisible or visible (Note: make it visible only to better visualize the agent’s behavior). As for the platform, it can be resized at will, even making it rectangular shaped, and can also be translated (not rotated).
Regarding the 'Game' tab that can be viewed after pressing 'Play,' it is important to note that you can switch between the top-down view and the third-person view by changing from 'Display' 1 to 2.



Display 1



Display 2

3 Final Consideration

In conclusion, the project successfully adheres to the specified requirements, hopefully demonstrating a well-engineered approach to software development and a conscious decision making regarding boundary cases.
This endeavor has provided a valuable opportunity to familiarize myself with the Unity game engine, allowing me to confront challenges such as machine calculation errors and the implementation of a complex behavior for an NPC.
Navigating through these difficulties has not only expanded my technical skills but also deepened my understanding of game development principles. This project will certainly prove to be a significant milestone of my studies.