REPORT ON THE VERLET INTEGRATOR

Introduction

The aim of this report is to explain the Aim Bot we developed.

However, before we start explaining our goals as a team, we should explain our initial goals as a the bigger group we had before, since we had to scope down the project due to the team reduction.

Before starting to talk about the project itself, it is important to explain the goals of the previous group, where we 7 people. That project did not work out in the end, but since this one is based on the other, it is important to mention them. We divided the project in 4 main chunks of work. The first part was about doing the research on the Montecarlo and stating in the report the functions we were going to use, what our code was going to be based on. The second part was about doing the core, the base code; this means putting together the Verlet Integrator and the Montecarlo together to get the rough base of the Aim Bot. The third part was about designing and developing an interface, which we wanted it to be interactable, to represent with graphics the movement. The last part was about using the Montecarlo in a small video game so the user could see the use of it and how it would work in the “real world”. Realistically, our aim was to finish the interface and, if we had time left, implement the game.

However, as stated before, this project didn’t come through and the project was restarted. But it had to be scoped down since now we only had 3 people working on it when before we were 7. In addition, we had to restart the project with less time before the deadline since we started working later on.

Therefore, our main goal for this project is to get a Montecarlo well implemented in the Aim Bot with an interface. The user will be able to change the the input through an .xml file. The input is going to be the position of the bullet, the position of the target, if the collision is elastic or completely inelastic and the mass of the bullet. The program is not intended to have a rerun option, but if there is time, it will be considered. The Aim Bot won’t freeze the screen, so if the target is not found it will throw the bullet at a set speed and angle and notify the user the object has not been found.

The Montecarlo method uses a loop inside a loop. In the internal loop, the function runs for X times a random angle and velocity and computes for each one if the projectile reached the objective using a function, which in our case it will be called “PropagateAll()”, and the Verlet Integrator. In the external loop, this loop is limited to frames so the screen doesn’t freeze. This is the concrete case for video games. Basically, in general, the MC relies on a repeated random sampling (we require a sequence of numbers which are random, independent, real and uniformly distributed in the range zero to one) when it’s difficult to impossible to use other more deterministic approaches. It is used in optimization, integration and probability distribution.

In the following web pages it is possible to find more information on the topic:
https://en.wikipedia.org/wiki/Monte_Carlo_method
https://arxiv.org/pdf/cond-mat/0104167.pdf
http://itf.fys.kuleuven.be/~fpspXIII/material/Barkema_FPSPXIII.pdf
https://www.mv.helsinki.fi/home/rummukai/lectures/montecarlo_oulu/lectures/mc_notes1.pdf
https://towardsdatascience.com/the-house-always-wins-monte-carlo-simulation-eb82787da2a3
https://www.palisade.com/risk/monte_carlo_simulation.asp
https://www.sciencedirect.com/topics/neuroscience/monte-carlo-method

An Aim Bot, which is what we are going to use the Montecarlo for, is a “cheat” to never miss the target in video games, usually in shooters, since it is used to automatically aim and shoot. However, in this case, bullets can get to the target with curves, so it is easy to detect and report in games. It is also used to help the user improve their aim, not only as a cheat to win, as an assistant when shooting to help them learn. In our case, we are going to implement an Aim Bot similar to Worms’. Just the computer to learn and show how they work. Therefore, it’s going to be only the bullet and target and the program trying to find and hit the object, using Montecarlo and Verlet, the first one to compute the randomized speed and angle of the starting conditions of the bullet and the second one to calculate the phisics of how the bullet would act before hitting the object.

In the following web pages it is possible to find more information on the topic:
https://www.geekno.com/glosario/aimbot
https://www.urbandictionary.com/define.php?term=Aimbot
http://www.gamerdic.es/termino/aimbot/
https://en.wikipedia.org/wiki/Cheating_in_online_games#Aimbots_and_triggerbots
https://en.wikipedia.org/wiki/Cheating_in_video_games

The Montecarlo, since it is a method, it does not use formulas. What it does is set the velocity and the angle randomly in between some parameters to later implement the Verlet Method to calculate if the bullet hits the objective or not. The velocity goes from 0 to 700 pixels/s, since from trying out it was the most suitable choice. When it comes to angle, it can go from 0 to 180º, since it can go backwards this way and the

180-360 range is not needed to reach the target since the bullet can arch and fall down. The Verlet formulas can be found in our previous report on our previous project, found in the repository “PHYSICS2-Verlet_Integrator”.
Repository “PHYSICS2-Verlet_Integrator”:
https://github.com/Needlesslord/PHYSICS2-Verlet_Integrator

For more information, please refer to the links stated before. In addition, in our GitHub there is a folder called “Data” where you can find a Verlet Integrator done in MatLab and an example of an Aim Bot, which belong to David de la Torre.
Folder “Data”:
https://github.com/Needlesslord/PHYSICS2-Aim_Bot_SS/tree/master/Data

To summarize, our intention is to implement our previous Verlet Integrator with a Montecarlo method to get a bullet to hit a target. The possible inputs, which will be inputted from an .xml, are going to be the initial position of the bullet, the position of the target, the mass of the bullet, if the collision is elastic or completely inelastic and the size (radius) of the target. If there is time, we will implement a reload option to avoid exiting the app.

Implementation

Our Aim Bot, as stated before in the introduction, calculates if a ball can hit another ball. It uses Montecarlo to try almost every speed-angle combination and, if it hits the target, it saves the data, which will be the one represented. There is a 10-loop iteration of the Montecarlo, which iterates the Verlet 1000 times.

We decided to use xml to input the values in real time because it’s a useful tool that allows the user to specify the concrete values they desire to see in the simulation instead of using keys to move the elements, which could be more intuitive and practical but wouldn’t be as precise.

We used the Montecarlo Method, which consists of checking the trajectory of the bullet using random numbers within a desired range for the initial speed and angle of the ball until a trajectory that coincides with the objective’s position is found. To simulate each trajectory we made use of the Verlet Integrator we made for the first project. In order not to freeze the Game Loop, the process is stopped every 1000 iterations even if the right values aren’t found and continues in the next loop. To control the amount of frames, we did a loop of 10 iterations of the Montecarlo after trying out what it usually needed to compute the result. Vsync was used to align the logic with the visual representation.

For the base code, we created the simulation with a console. We decided to wait instead of asking in the console for the data since we knew we were going to try to implement an interface. There is, however, a folder in our GitHub where the code from that is visible.
Folder "AimBot - Console":
https://github.com/Needlesslord/PHYSICS2-Aim_Bot_SS/tree/master/AimBot%20-%20Console

Once we had the base code, we started implementing the interface, which is thoroughly explained below. It works in the same way but the output changes from the console, which is text, to a graphic representation. You can also find the code in the folder "AimBot - Interface".
Folder "AimBot - Interface":
https://github.com/Needlesslord/PHYSICS2-Aim_Bot_SS/tree/master/AimBot%20-%20Interface

The inputs are managed from an .xml so the app can reload different data without exiting. There are actually 2 .xml's, one called "config.xml", which is used for the configuration of the interface and should not be touched, and another one called "input.xml", where the user can change the data of operation. The data they can change is the position of the target and its edge (radius) and the initial position of the bullet. In addition, the "Enter"/"Return" key is used to start the representation of the the ball and the "R" key is used to reload the information of "input.xml" the user may have changed.

The output is the visual representation of the bullet hitting/missing the target. In addition, if it has been hit it will blit a text saying "HIT!" and if it misses it will say "MISS!".

We used the object class, which is the way in which we already implemented the Verlet Integrator, to define the bullet and the target and to give them all of their values for the Integrator to compute and for xml files to input. In total, there are two objects, the bullet and the target.
When the program is started, the AimBot process is started too and it stores the values found as soon as possible.
When 'Enter' is pressed, the representation of the movement of the bullet is visualized until it reaches its destination, meaning the trajectory is propagated and also printed on screen this time. If no right values were found, the bullet will just fall.
When 'R' is pressed, the information stored in input.xml is loaded into the simulation and the AimBot is run again, prepared for the next time the visual representation is started.

The interface is very simple and based on Dragon Ball, since our team's name is Vegetta Super Saiyan, Vegetta SS for short, as well as all the other teams's. The sprites are inpixel art, with a Dragon Ball as the bullet, a Vegetta Super Saian as the initial position of the bullet, Boo as the target and a background of the dojo. In addition, with a timer, when Vegetta "throws" the ball, his leg bends as if he was kicking it, but we decided not to do an animation since we thought it matched the simple design and it was funny. In addition, it has background music of Dragon Ball.

The interface, however, does not show the initial velocity of the bullet. In case you want to know which is the angle and the initial velocity, please check out the release of the console version of the Aim Bot.
Release v0.5:
https://github.com/Needlesslord/PHYSICS2-Aim_Bot_SS/releases/tag/v0.5

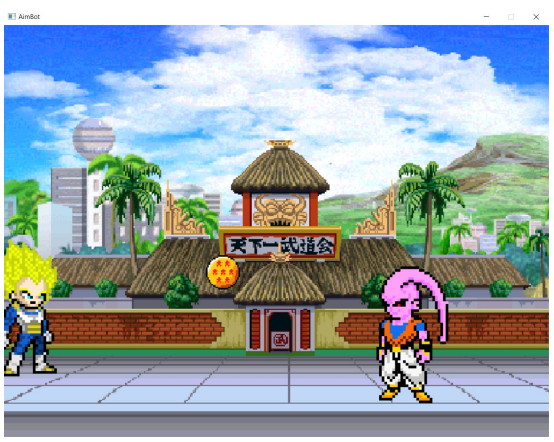
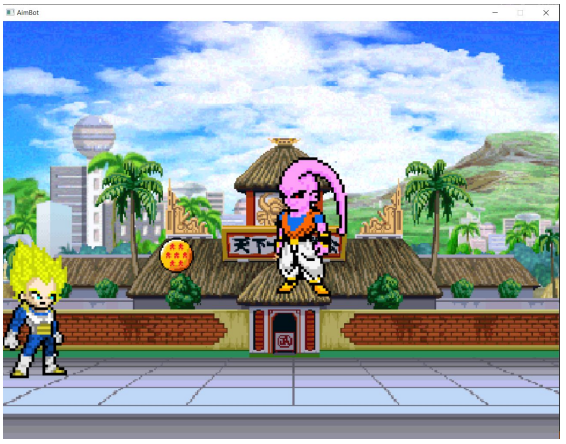
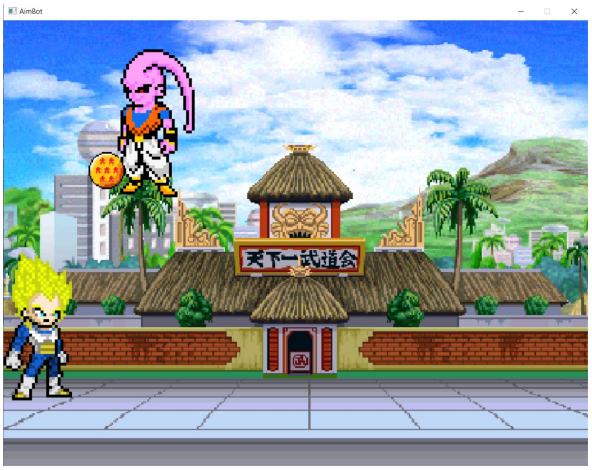
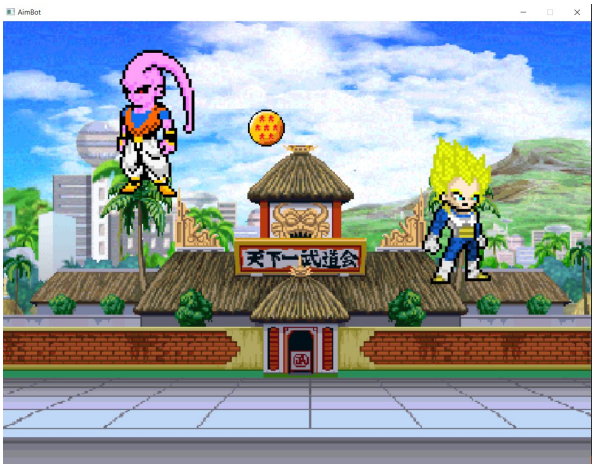
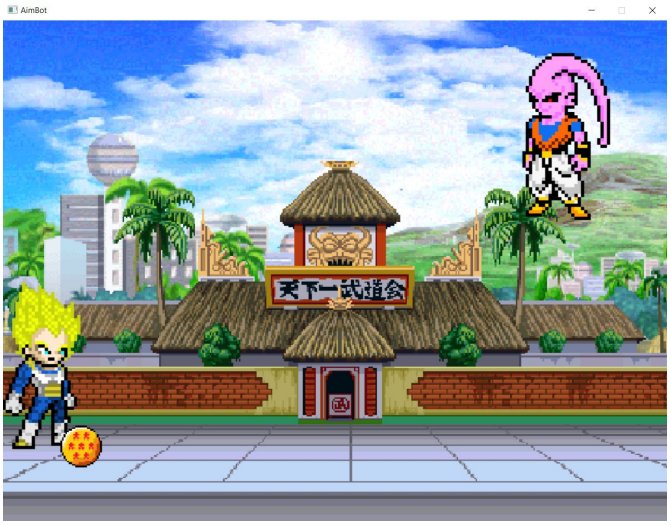
We accomplished everything we wanted and a little bit more. As said in the introduction, we wanted to develop an Aim Bot with an interface, which we did. The Montecarlo works perfectly with the Verlet, the screen doesn't freeze and the the movement is done smoothly. We implemented as an extra the keys "Enter"/"Return" and "R", the first one to shoot and the second one to load the info the user wants from an .xml. In the end, we decided to no let the user choose whether the collision was elastic or not because the can exit and reenter the window if needed, instead of colliding with the limits. We thought it was more realistic and unnecessary. We are confident in our Aim Bot and happy and proud of the results, even with the reduced scope.

Results

The tests we are going to run are the following:
(1) That the code does not crash in any way during the simulation or the different steps.
(2) All data set to 0.
(3) 5 different simulations with random numbers.

(1) The app did not crash indeed in any of the simulations or when pressing any key or changing the input.
(2) We designed safety checks so the simulation didn't crash. Even if it in theory possible in physics, we wanted more the feel of a game, where you will never have, for example, a negative edge or 0 length. Therefore, if the positions are set to 0, they will be in a corner and if the edge is 0 or less it won't be updated.

(3) The 5 random simulations worked perfectly, whether the target was found or not. The following screenshots show the simulations with the ball midways. They all found the objective, except for the last one, which was tested to check if the game crashed if the Aim Bot missed.

Comparison with other Aim Bots: Worms

What the Worms AI's does is aim directly at the other team's worms, normally at the closest one. The weapon is more or less random, but it is clever enough to decide not to use an Air Attack when they are literally next to each other. In addition, the game has collisions/obstacles in between the shooter and the target so there are fewer ways to reach the objective. Once it has found the objective and chosen the weapon, it calculates the angle and the power of the shot before shooting. If the objective is not found, it shoots randomly.

In our case, however, we do not have obstacles in the scene and therefore it is easier to find the target. In addition, we only have one way to throw the ball, so it is not needed to choose a weapon. Moreover, we only have one target, which is also easier and does not require the Aim Bot to decide who to shoot.

To summarize, the Worms' IA is more complex than ours, but it is expected since they have a very developed game whereas we only have a simple Aim Bot with one way of shooting and only one target. However, overall all the needs are met in both, just that in ours the scope is smaller.

Conclusions

Our Aim Bot works perfectly in both console form and with interface. The test in the results show that it is trustworthy and that it works well. In addition, the simulation works smoothly and the screen does not freeze; therefore it would be usable in a game. We encourage you to try it and use it freely.

by:
Tomás Carreras
Enric-G. Durán
Núria Lamonja (Team Leader)

GitHub Repository:
https://github.com/Needlesslord/PHYSICS2-Aim_Bot_SS

Thanks to:
David de la Torre

Project done for:
CITM-TTC