Introducing AI Material to CS 4730 – Game Design

A Capstone Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Karim Shoorbajee

May 10, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Karim Shoorbajee

Capstone advisor: Madhav Marathe, Department of Computer Science

Introducing AI Material to CS4730 – Game Design

Karim Shoorbajee
Computer Science
University of Virginia
ks4fd@virginia.edu

Abstract

Modern video games often include non-playable characters and opponents controlled by Artificial Intelligence. This paper proposes an AI unit to be added to CS 4730: Computer Game Design. It explains what AI is in games and covers things like enemy/npc AI. The proposal includes a week's worth of material to be taught in this unit and how it will fit into the course as it is. It also includes examples of homework based in the Unity game engine that would be completed alongside the instruction. Well-programmed AI makes games more balanced, immersive, and enjoyable. Modern games are utilizing increasingly complex and believable AIs so an AI unit is of great importance to a game design class.

Introduction

There is minimal discussion about AI in UVA's current game design class. Some form of enemy or opponent behavior is implemented in games across several genres. Specifically, path finding is used in popular games to program enemies to follow the player. Playing a game where enemies make strange decisions and do not react to player behavior properly can hinder game aesthetics. These aesthetics can include challenge, fantasy, and sensation. It is very important for students to learn about more advanced enemy design techniques in order to gain a more wholesome understanding and appreciation for game design.

There is discussion about enemy design currently in CS 4730 but the discussion focuses on state-machine level behavior definitions for non-playable characters (NPCs) and enemies. State-machine defined behavior is robust and useful but does not go in depth enough to allow for interesting and practical enemy behavior. This is why AI topics like efficient search / path finding should be introduced into the course material.

This paper proposes an additional AI unit for CS 4730 Game Design where students will learn about path finding algorithms used in modern video games. The proposal will also include assignment description and a sample assignment where students can practice implementation of these principles.

Background

There is game design and AI specific domain knowledge involved in this project. For game design, I based my introduction of this material primarily because of the Mechanics Dynamics Aesthetics structure[1] (MDA). MDA defines interaction between the player and the game's programmer. The player first experiences a game's aesthetics (how the game makes the player feel on an emotional level). The aesthetics are derived from combined game dynamics. Dynamics are game rules that combine over a time to create experiences. Dynamics might include how a player needs to repeatedly do an action in order to level up their character. Another example could be playing a sports game and selecting a tactic that makes the player's team play in a certain style. Lastly, mechanics are code level rules in the game. For instance, pressing a certain button causes the player to jump. Another example might be moving the player close to a tree and pressing a button causes the player to chop down the tree. In general, the player is presented the game in the order of Aesthetics, Dynamics, and Mechanics while the game programmer builds the game in the reverse order. These are relevant to the justification of building a robust path finding system.

Also relevant to the game design domain knowledge: Unity was used to implement the homework assignment. Unity is a game engine that abstracts common game design tools to allow programmers to focus on more high level aspects of games programming. For instance unity handles the game loop for any element in the game. There are also out-of-the-box collision detection, events systems, and physics that programmers can use. It also provides a visual representation of the game for programmers to easily drag and resize specific game elements.

The AI topic most important to this project is A*. A* is a path finding algorithm that many modern games use. A* is an extension of Dijkstra's algorithm, and also computes a shortest path from given a weighted graph. The advantage of A* is that it prioritizes searching in the direction of a specific goal node. In an A* implementation, the queue will prioritize based on the lowest distance traveled + distance to the goal node. Dijkstra's will search indiscriminately, in all directions simply picking the shortest distance traveled so far while A* will prioritize searching towards the goal, which is more efficient in the

context of game design since we often know what a certain NPC or enemy should be searching towards. The AI principles presented in this proposal are based on material from CS 4710 Artificial Intelligence.

Related Work

Since this additional material will be added to an existing course, we need to identify material that could be removed to make time in the schedule. One homework assignment and one week of lecture material should be removed. As for homework, the game loop assignment should be removed. The game loop asks for the students to develop a simple text based game in python that runs in the command line. The argument for having the game loop as a homework assignment is that it presents an environment familiar to upper level computer science students to dip their toes into game design concepts. The game loop explores games having objectives, conflict, and outcome as discussed in the Formal Elements of Games lecture, since the game loop has to have some sort of dialogue that directs the player to make decisions and reach an outcome. This is obviously valuable to the game design course. However, the semester-long project where we create a game in a group will already cover this material. It can be slightly redundant to exercise this material twice when there are other interesting topics in game design to be discussed like artificial intelligence in games.

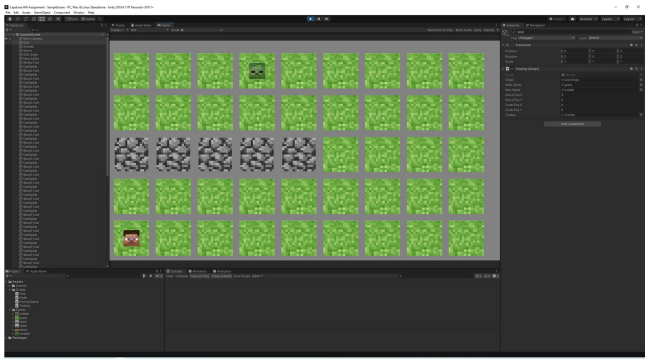
For lecture material, the History of Games should be removed. The justification for this is that the material does not carry relevance to the implementation of the semester long project or any of the other individual homework assignments. Much of the material covers the different generations of video game consoles and how much revenue they generated. It also covers different marketing strategies employed by console manufacturers. The history of video games is very interesting but I would argue a unit on artificial intelligence in games would be more beneficial. This is not to say that all lectures in CS4730 Game Design should have direct relevance to the implementation level of homework assignments. For instance the lecture titled Actions and Interactions doesn't directly relate to the programming done in games but a lot of the concepts can influence decisions made in the design process. It can help students understand how user inputs map to specific actions and how they should combine and interact with the game. A similar line of reasoning cannot be drawn out of the History of Games lecture and thus should be removed to accommodate an artificial intelligence section.

System Design

Determining coursework to be removed. In order to determine how to make room for the added AI material, I referred to my coursework from the game design class and assessed their relevance to the course. This involved reading through course slide sets and looking at course homework/projects. I assessed which of these were of lowest importance or relevance to the class and suggested their removal accordingly.

Making the slide set. I determined the best way for the material to be taught is using a slide set. This is traditionally how students are lectured in the game design class. Usually the slide sets are presented alongside a lecture from an instructor. The slides do not necessarily stand on their own for students to learn from. The intention is to summarize important points on the slide set and leave the finer details to be lectured and for students to ask questions in class. Slides should be more visually engaging. This is why I also included visual representations of Dijkstra's and A* algorithms running on a grid. The slide set is also where I connected the importance of believable AI back to the MDA structure, arguing that good AI is important to the aesthetic of fantasy.

Designing the Unity Assignment. The assignment for students to complete alongside the lecture material was developed in the Unity Game engine. The assignment is a 2D unity project which students would download and come preconfigured for the students to implement A* and verify that it is functioning properly.



Screenshot of the Unity Project's game view

When opening the project, the students are presented with a game screen that is a grid with two characters placed on cells of the grid. The grid will be the search space for the path finding algorithm. The grid by default will be set so that all the cells are traversable, indicated by the grass sprite (graphic) on each cell. When starting the game the students will be able to left click on any of the cells to convert them to non-traversable cell, indicated by a stone sprite. They can left click again on the cell to convert it back to a walkable cell. The two characters are the monster and the player. The end goal of the assignment is implement A* so that the monster will travel to the player's position while only traversing on walkable cells. Player is synonymous with goal or end in an A* implementation. The monster represents the

starting node in an A* search. The cells on which the player and the character start are walkable by default and cannot be changed by left clicking. The default starting position of the player and the monster can be changed in the main testing script of the project. The distances in this grid are measured are considered Chebyshev distances, i.e. traveling one cell in the cardinal directions or diagonally is considered a travel distance of 1. The grid is considered to be un-weighted. Each node is connected in all 8 directions unless there is a stone cell in an adjacent cell.

There are 3 primary C# scripts that run the Unity project. Grid.cs is the script that contains the implementation of the grid that is drawn on the screen. The grid implementation is based on an implementation from a Unity tutorial[3]. From here, one can decide a width and height of the grid, size of a cell, and the sprites used for traversable and non-traversable cells. A grid object controls the primary behaviors of the grid. It enables the student to set and unset traversable cells. It offers many helper functions for the player to convert the position of a mouse click to a position on the grid coordinate system and vice-versa. It also allows for the student to get the position of the player which is crucial for the implementation of A*. When the student starts the homework assignment, the Grid class will contain the unimplemented method called *chase()*. This will be where they implement the A* algorithm given the supplied helper methods and grid system.

Node.cs contains an un implemented class that will represent a cell in the grid in an A* implementation. The node class is where students will implement the ordering priority in a priority queue. Its constructor would also automatically determine its cost in the search by adding the distance travelled to get to the cell to the distance of the cell from the player. The student will use the node class in their implementation of the chase method.

The third script relevant to the implementation is the Testing.cs C# script. This script inherits the MonoBehavior class from the Unity engine, meaning it includes a game loop which will be run on each frame of the game. It is attached to an empty game object in the world. This is where an instance of the grid class is defined and drawn into the world. The chase function in the grid class returns a list of cells for the monster to travel through to reach the player. When the student launches the game and left clicks to build a path through which the monster must travel, they can then right click anywhere on the screen and have the monster traverse the path returned by the chase method.

After the chase method is called in the testing script, the monster will traverse whatever path is returned from the chase method. Students will then be able to, by inspection determine if their A* search is functioning properly. The monster will travel from cell to cell in the order of coordinates returned from the chase function.

A priority queue is an important aspect of an A* implementation as it allows us to determine the next lowest weight node to travel to without maintaining a strict ordering of queued nodes. C#, which is the scripting language used in the Unity engine, does not have an included priority queue implementation in its libraries. The implementation of a priority queue is out of the scope of a game design class, and students will have learned how to implement on in CS 2150, which is a prerequisite for this class. Having the students implement their own would be unnecessary. Thus, I opted to use an external implementation of a priority queue in C#[2].

Determining the Responsibility of the Student. An important design decision when creating a homework assignment is determining what level of implementation is the responsibility of the instructor. Essentially, what is considered 'boilerplate' code. This has to do with the objective of the assignment. The implementation of the grid system in this assignment would be within the domain knowledge of the students assuming this assignment would be presented in a later part of the course. In the game design class, we learn about manipulating and constructing objects in the game world, and have them be dynamic while the game is being played. However, I decided this should not be expected of the student for a path finding assignment. There is nothing novel about this assignments specific implementation in the Unity engine. The novelty comes from the A* implementation. Asking students to implement the grid system for themselves would be redundant, time consuming, and of little educational value.

An important part of the A* algorithm is how to prioritize nodes to visit. Implementation for this exists in the node class. In the Node class we override the comparison method to prioritize nodes based on the A* heuristic. The node class also stores the optimal node from which we reach the current node in order to back-track the optimal path. Because the Node class is integral to A*, it's important for the students to implement it themselves. Implementing the chase method and the Node class is sufficient for the students to understand the core concepts of the new material. It also does not reteach material that is covered in other sections of the course.

Procedure

In order to complete the assignment, students will download the unity project described in the system design section from the course website. They will open the project in Unity and open the scripts to implement A*. They would use the game view to test if their A* algorithm is implemented properly. If they draw any path that allows the monster to reach the player, the monster should reach them. The assignment would then be submitted for instructors to grade.

Results

The homework assignment was presented to prior game design students. The example solution was provided. The general consensus amongst the users was that this would be a valuable addition to the game design class. They accounted that this would open up possibilities in the design of the semester long project. They found that after a brief explanation, the given unity project was easy to navigate, and it would be easy for a student to understand the required task. They found the path building system to be intuitive. They also stated that testing via the monster being animated to follow the player was sufficient, to determine if the path finding was working properly. They noted that AI and specifically path finding is ubiquitous in modern game design which makes the addition of such a unit more valuable.

The prior game design students were also shown the slide deck to be taught prior to the implementation of this assignment. The consensus was that it is adequate for the students to be lectured from, giving them enough information to implement A* afterwards.

User Suggestions. Users offered valuable feedback and proposed additions that could improve the assignment. They suggested that this material and homework assignment should be introduced as early as possible. Learning about A* path finding enables students to envision top-down 2D games with robust path finding. This could influence the entire structure of a student's semester long project so learning about it early gives them ample time to build around the concept.

It was my intention to abstract the Unity specific programming out of this project and focus primarily on the implementation of A*. A user suggested that learning about how the grid system works in unity would also be valuable, since this is the scaffolding over which the algorithm is implemented. In order to visualize how students might utilize path finding in their own games, it would be beneficial to implement the grid system themselves. This reasoning is valid and I do see the value of the students implementing their own grid system. But the amount of time required by students to implement both the grid system and the path finding algorithm would likely be longer than a week. Other homework assignments would have to be shortened to accommodate this change. Also, as stated before, the grid system is not out of the scope of what is already taught in the class, and the grid code is provided for the students to learn from and reimplement.

Another user mentioned that the monster is not animated in chasing the player. The monster simply moves from one cell to the next in a single frame. They mentioned that, for extra credit, a student could animate the monster to make the project more presentable and akin to an actual game. This would involve the student modifying some of the code in the testing script. This would be in line with other extra credit / optional points offered in the current game design class.

Users also noticed that the testing was primarily by inspection. The students themselves have to determine if the path followed by the monster was the shortest path in Chebyshev distance. The user suggested that this might not be completely intuitive to a student having their first encounter with A*. They suggested adding a way of detecting that the monster traversed multiple cells in one move, or if the monster traversed a non-walkable cell. This would be trivial to implement as an instructor. It would involve assessing the list returned from the chase method and checking if each move is valid (only a distance of one) and if each cell is walkable.

Conclusions

This project was a culmination of CS 4710 Artificial Intelligence and CS 4730 Game Design. The purpose was to propose a unit for CS 4730 Game Design that incorporates topics from AI. The methodology was to provide a sample of material to be lectured and a corresponding homework assignment. The focus was on the AI topic of local search, specifically the A* algorithm. The lecture material was to be applied to a homework assignment also designed as part of this project. The homework assignment consists of a Unity project providing scaffolding and boilerplate code for students to implement A*. The students would test their implementation in the Unity project using the provided game assets.

The problem this project was addressing was a lack of believable and intelligent enemy and NPC behavior taught as part of the game design class. I sought to use concepts from AI to enrich and enhance learning in the game design class. The game design concepts I was seeking to enhance were directly related to MDA, and how the user perceives games.

The project was successful in adding a new paradigm to the game design class as was reported by the students to whom I presented the system. They noted that this would have been valuable information to learn and apply to a semester long project. They also noted the relevance of AI to modern game design.

Future work

There is much potential for additional AI materials to be added to game design. I did not cover the basis for AI in tabletop games like chess. There could be more material added to the game about designing AIs to play against human players. For instance, we could design a simple tic-tac-toe playing AI, as this is discussed in an AI lecture. There could be material discussing Alpha-Beta pruning, which allows for more efficient searching of trees that represent game states in games like tic-tac-toe or chess.

An important thing to discuss if adding material on AIs playing human players is how to maintain good game design while doing so. No player wants to always play against a tic-tac-toe game that plays perfectly every game. Game balancing, is discussed in game design and should be accounted for when discussing AIs in table top games. Students should also be expected to make any AI that plays against a human decently challenging but not impossible to beat.

REFERENCES

[1] Robin Hunicke, Marc LeBlanc, Robert Zubek,. MDA: A formal Approach to Game Design and Game Research https://users.cs.northwestern.edu/~hunicke/pubs/MDA.pdf

[2] James McCaffrey. 2012. Priority Queues With C#. Visual Studio Magazine. https://visualstudiomagazine.com/Articles/2012/11/01/Priority-Queues-with-C.aspx?Page=1

[3] Code Monkey. 2019. Grid System in Unity (Heatmap, Pathfinding, Building Area) https://www.youtube.com/watch?v=waEsGu--9P8