Artificial Intelligence for Games – Project Research Workbook

This workbook will help you focus your research for your project.  
Once you have answered these questions, use this information in your Technical Design Document.

|  |
| --- |
| **Briefly describe the application, game or simulation you are researching.**  **(This is your initial idea to focus your research. The application described in your design documents or your final build may end up being different from this description)** |
| The simulation I am aiming to create is a maze solving simulator.  The idea of this project will be to take in an image of a maze that the user will either draw or upload externally as a png.  The user can then left click and right click to set start and end points of the maze.  Once the image is imported or the maze is drawn, the user will have the option of either stepping forward to watch the pathfinding algorithm solve the maze one step at a time; or instantly let it solve the maze.  Information will be given to the user such as steps required, maze size and solve speed.  The path will be instantly calculated instantly, but a little character will be used to represent this path for each step. This will be the agent  The character will then take a step each x number of seconds, which may be modifiable by the user.  The user can also use the toolset provided to create and edit their own maze on the fly, as well as generate a maze with a certain percentage of walkable space. |

|  |
| --- |
| **What pathfinding algorithm will you implement.**  **List some (at least one) online or class resources that will assist you in implementing this algorithm.**  **Examples may include YouTube videos, blogs, textbooks, or class resources** |
| A\* is the most suitable algorithm for this project.  <https://www.youtube.com/watch?v=-L-WgKMFuhE&t=6s> and in class lectures will be extremely helpful in implementing and understanding the algorithm.  The reason A\* is the most suitable is due to how flexible it can be with the pathfinding. Since there will always be one start position and one end position, this makes A\* the best choice as it does not waste time and resources checking every single node.  It is also easily modifiable. If for whatever reason the user decides to change the base costs of nodes, they can. This will result in the pathfinding AI solving certain setups differently. An example of this is straight up avoiding diagonal movement. |

|  |
| --- |
| **Describe how this pathfinding algorithm will be used by the agents in your application.**  **For example,**   * **Will the player-controlled character find a path to the last click?** * **Will AI agents follow a target?** * **Will the destination be updated dynamically (if so, at what interval), or only when the agent reaches its current destination?**   **Reflect on (and describe here) how these choices influence the design and development of your application.** |
| The agent will be player controlled, as the player must set a start and end point. The agent will then move every time the user decides to step forward. This of course, uses the A\* path that is calculated behind the scenes. |

|  |
| --- |
| **What AI strategies could be used. (For example, state-machines, blackboards, decision trees, etc)**  **Pick two strategies and list a brief description of how they work, along with some resources that could assist you in implementing these strategies. (Examples may include YouTube videos, blogs, textbooks, or class resources)** |
| State-Machines might be used, since the agent may need to have a its state changed whenever it makes a step.  State machines simply hold a state that are typically represented as an Enum. For the sake of this project, each state may have a different update function. This makes managing the AI and controlling it programmatically a lot easier.  Some states could be MOVING\_NEXT\_NODE and WAITING\_INPUT. |

|  |
| --- |
| **What is a Technical Design Document, and why is it useful?**  **List the major topics or points of discussion (at least 5) commonly included in technical design documents.** |
| The major topics to be included in a Technical Design Document are:   1. **Product requirements are defined**. These will typically be represented by a Product Requirements Document (PRD). The PRD specifies what the system needs to do, from the perspective of a user or outside agent. 2. **Technical requirements are defined**. The product requirements are translated into technical requirements — *what* the system needs to accomplish, but now *how* it does it. The output of this step is a Technical Requirements Document (TRD). 3. **Technical design**. This contains a technical description of the solution to the requirements outlined in the previous steps. The TDD is the output of this step. 4. **Implementation**. This is the stage where the solution is actually built. 5. **Testing**. The system is tested against the PRD and TRD to ensure that it actually fulfills the specified requirements. |

|  |
| --- |
| **List the libraries, APIs, algorithms, or assets (i.e., any pre-existing component) that you will use, or might consider using, when implementing your application.**  **For each component, include a brief statement listing it’s functionality, suitability, and technical impact on the project.** |
| A\* Algorithm  **Functionality –** Allows a path to be calculated from point a to point b(accounting walls)  **Suitability –** Will be suitable since it may be a maze solver; therefore, a clear path to the exit will be required.  **Impact on project –** Will be the main bulk of the project, as the agent will heavily rely on this algorithm to solve the maze.  Unity  **Functionality –** Allows visualisation of maze as well as user input.  **Suitability –** Will be suitable as shows a visualisation of the maze and user input.  **Impact on Project-** Will allow user to interact and view the project. |

|  |
| --- |
| **What are the licensing arrangements or restrictions for the pre-existing components you have identified?**  **(Identify the licence of each component)** |
| * Unity Standard Licence Agreement * Visual Studio Community Licence * AIE Student Agreement * MIT License A\* Algorithm |

|  |
| --- |
| **Identify a possible audience for your application, game or simulation.**  **What platform(s) are you targeting?** |
| A possible audience for this simulation application would be software designers and/or software engineers that wish to see a simple demonstration of the A\* algorithm in action.  The whole point of this project is to give the user a sandbox to experiment with A\* however they want. If they want, they may even implement new stuff within it to suit their needs. |

|  |
| --- |
| **Describe the real-world environment you will be simulating.**  **Dscribe any real-world aspects that are present in the game world, and how their real-world behaviour will influence your simulation.**  **For example, does your program have animals that will head towards water when thirsty? Or, will you implement prison guards that head towards the source of a noise?** |
| The project at the moment does not have any sort of real-world environments.  The only aspect which seems to be similar to a real-world environment would be the character avoiding walls when trying to reach the end point. |

|  |
| --- |
| **Try to describe the complexity of your project.**  **For example,**   * **How complex are the AI behaviours you have chosen?** * **How many classes are needed to simulate all your entities?** * **How deep is the inheritance hierarchy?** * **Which algorithms are the most complex or difficult to implement?** |
| This project in its current state is not necessarily overly complex.  The reason for this is since it is just implementing A\*, then implementing ways for the user to interact with the algorithm.  About 5 classes will be required:  **Visualizer** – In charge of visualizing the maze.  **Clicker** – In charge of user input(adding/removing walls as well as set start and end points.)  **Grid** – Container class that holds a 2D array of PathNode.  **PathNode** – Node which the maze is composed of.  **AStar** – The A\* algorithm written to adapt with PathNode and Grid classes. |

|  |
| --- |
| **Identify any tools that may help you in your implementation.**  **For example, graphical applications for asset development, debuggers or IDEs.** |
| Unity or RayLib will be extremely helpful applications to visualise this simulator. Since once the implementation is complete, it is just a matter of visualisation, which is not difficult at all.  Unity will most likely be the best choice since it is written in C# thus cutting work time. The lack of pointer management will also make it easier to port over the algorithm as well as Agent related classes. |

|  |
| --- |
| **List any other additional information that may aid in the design of your project.** |
|  |