Artificial Intelligence in User Interface and Experience.

David Petika and Leo Ryoo

# 1 Introduction

In 1997 Deep Blue, an AI developed by IBM, beat world-champion Garry Kasaprov at chess. And although this may not be the definitive moment, it was one of the first moments that showed the world that computers could be better than us at making decisions. Fast forward to today and any home computer, and even some smart phones, has the capability of playing chess as good as or better than deep blue. As such, game developers must ask themselves how to best use this potential in their games. The answer, typically, is to create AI that builds the world, or works against the player as opponents. But a seemingly unexplored possibility, and the purpose of this paper, is an AI that improves the player’s experience by predicting the player’s needs and guiding them toward their ideal experience.

# 2 The Project

To that affect, the goal of this project was to develop an AI that would communicate with the player through the user interface in order to improve that player’s experience. This entailed keeping track of the player’s likes and dislikes, and tailoring both the world and the UI’s guidance accordingly.

## 2.1 Research

Through research on the topic, it was learned that AI often played three roles when interfacing with user interface. The Advisor, which guides the player to the optimal outcome. The Curator, which observes the player’s preferences and changes its behavior accordingly. And lastly the Orchestrator, connecting with various other components of the game in order to produce the desired affect across the entire game.

Of course, before developing this AI we needed to first develop a game for the AI to exist in. For simplicities sake, the game chosen for this project was a game where the player was placed in a procedurally generated maze and had to find their way to the goal.

The AI would then advise the player about what direction they should take in the maze in order to get to the goal. Depending on where the player died, and what items they picked up along the way, the AI would then alter its advice to the player. connecting with other parts to produce a desired outcome. And once the goal was reached, AI would then pass off the data it acquired about the player to the procedural level generator, so that it could change the ratio of its rooms to Orchestrate a better player experience.

## 2.2 Advising

The first step, Advising, is relatively easy to implement because it’s the same kind of calculations you would do for an enemy’s AI, the only difference being that instead of connecting it to a game actor, you connect it to the user’s interface. For this project, the calculations were for pathfinding through the maze, but it could just as easily have been any other heuristic based calculation.

The important part of this step is that it should simply be advising the player, and not controlling the player. The AI always knows the best path but telling the player exactly what that path is removes the feeling of choice from the player which can ruin the experience. so instead the directions should be somewhat obfuscated to the player so that the player can still act on their own judgement. This balancing act between too much guidance is different for each project, but a good rule of thumb to use is simply “If the player could figure it out on their own, then don’t tell them to do it.”

In our example we’ve calculated a path from the spawn to the goal for the player to follow, there’s nothing else to direct the player towards the goal so the only way for them to know where to go is for us to communicate the calculated path to them properly. For the player to follow the path, they need to be told the next node in path to move to. The UI element for this needs to balance between the player being completely lost and giving the player so much instruction that they only move if the AI shows them where.

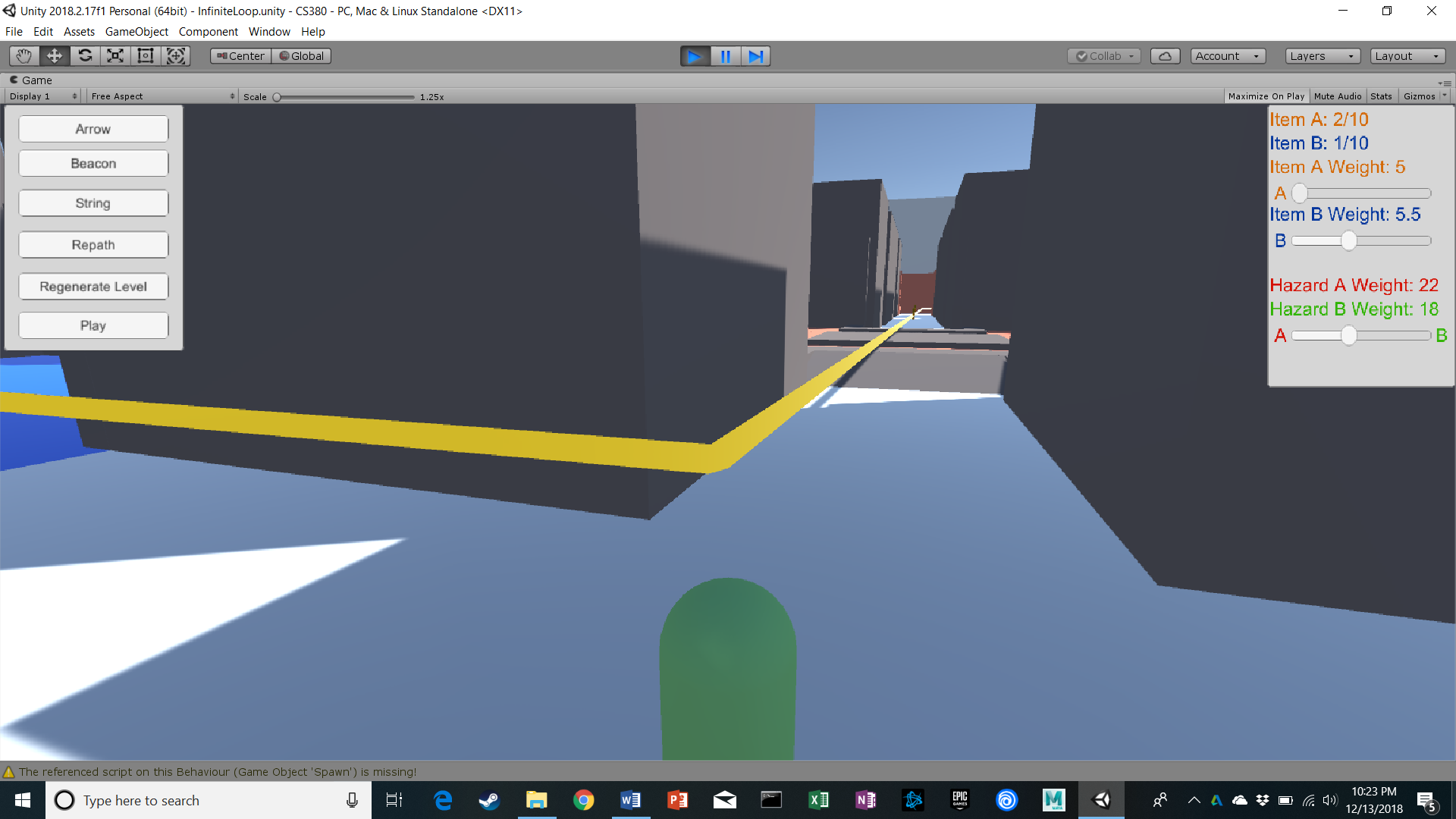


Figure 1

(Figure 1) If the path is shown as a solid line then we have the problem of not knowing which direction is forward and which is backward. But more importantly, the player will effectively just be moving down a given path without making decisions of their own.

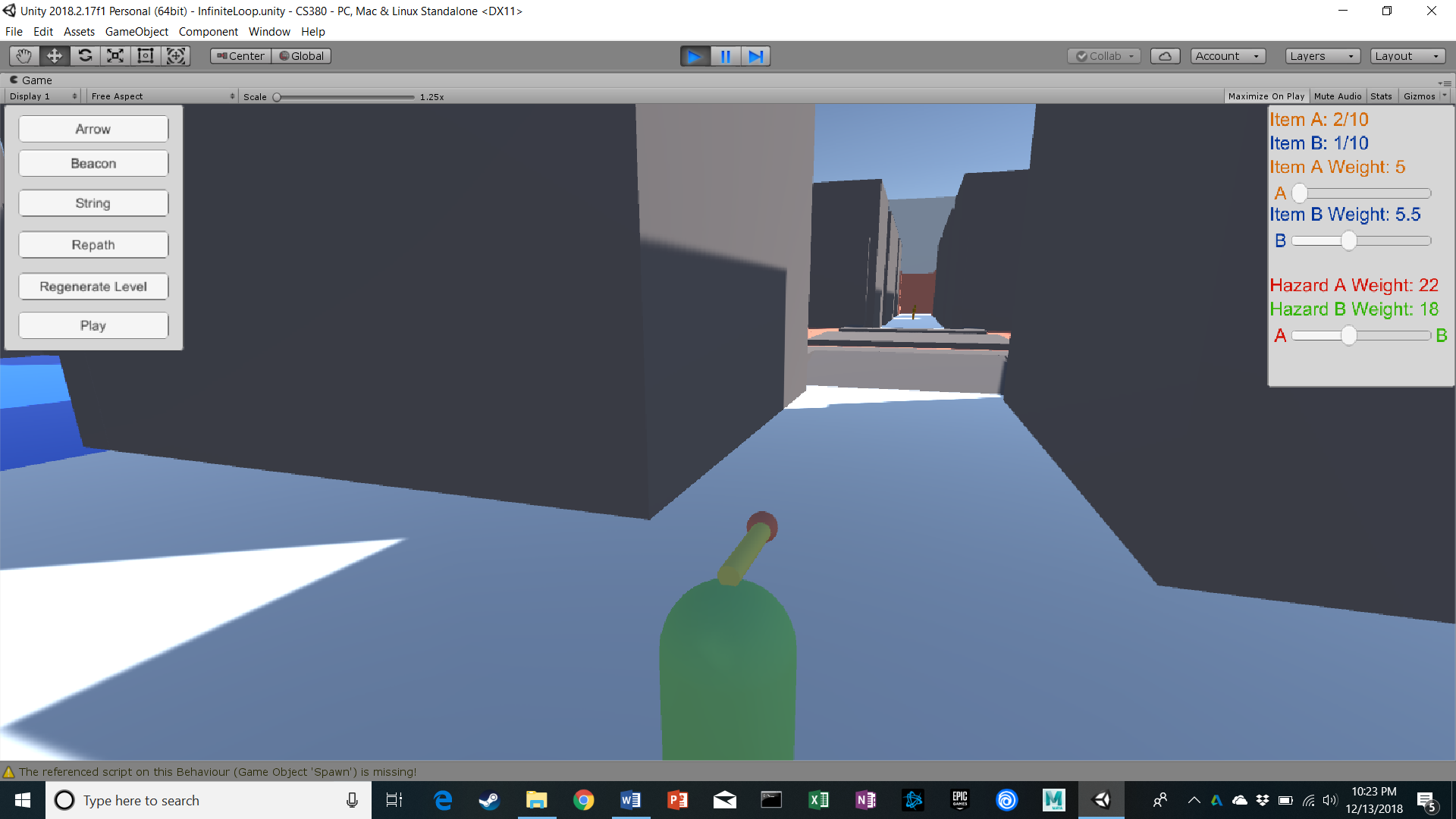


Figure 2

(Figure 2) If an arrow is shown pointing in the general direction of the next node in the path, then the player has freedom of deciding how they want to get to that point. However, because the player doesn’t know the exact location of the next node, they won’t have enough information to plan a path to the next node themselves, and instead will just move in whatever direction the arrow is pointing (making a similar problem to the line’s).

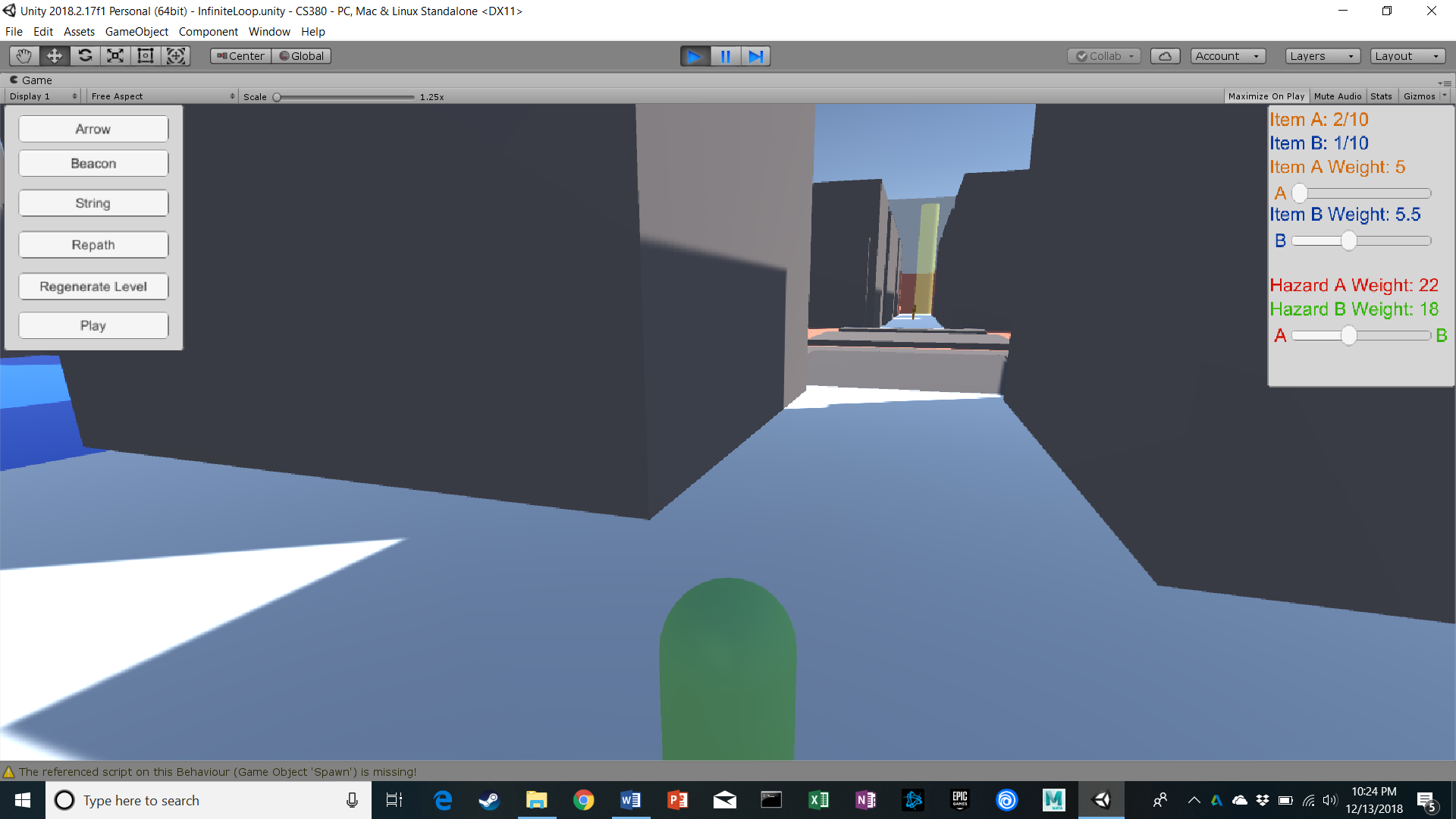


Figure 3

(Figure 3) If a beacon of light is added to the target node then the player has the current point and the next point to go to, so the player has enough information to make their own path towards the node. However, if the beacon of light is obstructed from the player’s line of sight then they wouldn’t know where to go.

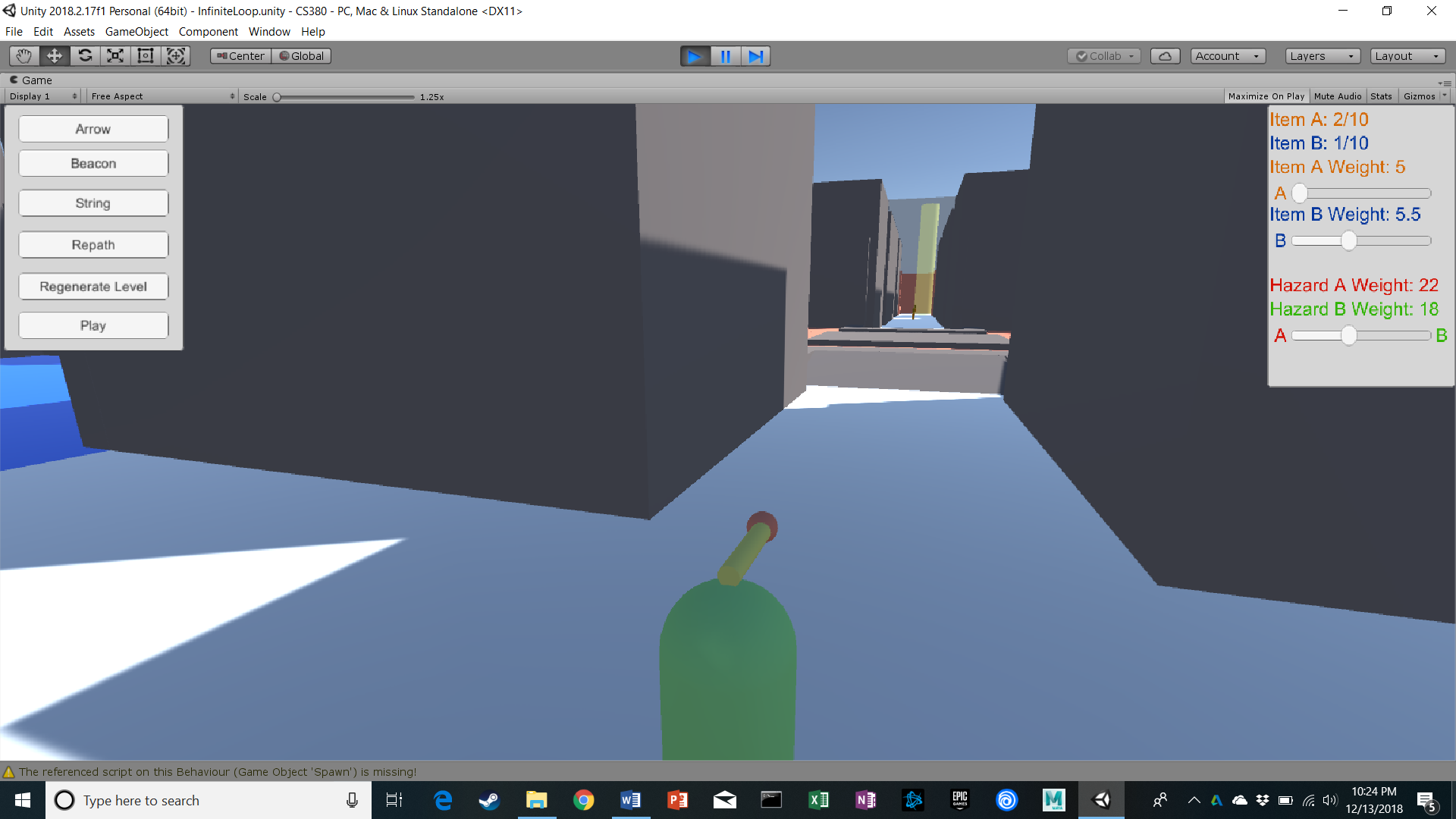


Figure 4

(Figure 4) Putting all the previous UI elements together, the “sweat spot” would be to use both the arrow and the beacon. Giving the player a start position, a goal position, and cardinal direction in case the player can’t see the goal position. This gives the player enough information for them to find their own path to the next node, but not enough information for them to not need to find their own path.

## 2.3 Curating

After advising the player, comes curating the player’s experience. This part proves to be the most labor-intensive part of the process because it requires you to make a player model to predict the player’s decision-making process. the data needed for the player model depends on what you plan on making the AI do, but for our project we wanted the path to change based on: how much of an item a player had already gotten, and how much they preferred a specific hazard.

Getting relevant data for the player model can get tricky because you’re not getting data on the player’s actions, but rather data on the player’s decision making process .This means that data like how many times a player dies to a given enemy is too simple, because it doesn’t show any connection between the players death to that kind of enemy, and the players decisions that lead to that outcome. Perhaps the player died to that kind of enemy because of an encounter where they must fight off five of that enemy at once? Or perhaps it’s because that enemy is only ever found in poison swamps, and the player isn’t managing the damage over time effects of the poison?

So, we can see that the process we need to undertake is more complicated than just counting the number of times player dies to a certain enemy or uses a certain object. The data we should actually be looking for is “given a set of decisions, which option does the player prefer?”, which is much more difficult to quantify. For our example, asking “which obstacle does the player die to the most to” doesn’t work because it doesn’t consider the frequency that the player the choice that the player makes. So, then the question becomes “which obstacle does the player choose more often, and how often does he die to that obstacle?” (1).

(1)

But even this equation isn’t perfect, because it doesn’t consider the factors that lead the player to make that choice. If your game has health points (2), or different equipment (3), then those factors would also influence the player’s decisions. The solution then is to calculate both how closely the player’s current state matches the state they’re usually in whenever they make the aforementioned decision, and the average amount by which the player deviates from that average. Once we’ve calculated all the values we need, all that’s left is to sum them together to get the current player’s heuristic for that option (4).

(1)

(2)

(3)

(4) \* (if ) exceeds 1.0 then you may want to cap it to 1.0 as to not exceed the maximum addition to cost)

Once the math is completed using the value is as simple as using area costs in the A\* algorithm: you present your AI with a start point, and a goal, and you give it various state changes with the weights that you calculated and allow it to compute the optimal path to the objective. For our project it’s a literal path calculation but you could just as easily apply the same concept to a calculation to find the player’s optimal inventory organization (get the heuristic importance of each item in the inventory at the current moment and order them from most to least important).

## 2.4 Orchestrating

After the curating phase, we enter the orchestration phase: the orchestrating phase is essentially the second half of the curating phase except at a macro scale. We take the stats calculated from the curator and we send them to various other parts of our game to be used to better adjust the experience. In our project we send these values to the level generator to weight the generation of rooms so that the rooms that the player has more trouble with are spawned closer to the goal, and the rooms the player find easier are spawned further away. Additionally.

Our implementation of the orchestrator acts as a counter balance to the curator’s actions: the curator helps the player avoid obstacles they struggle with, while the orchestrator spawns more of them. This is like the concept discussed earlier of not giving the player too much or too little help; if we’re letting the player avoid all the things that they struggle with then they’ll get bored faster than if we sprinkled in sudden encounters of things that they would find difficult.

That being said the orchestrator can be used for more than just a counter balance of the curator. For example, our project uses the orchestrator to also spawn more of the item that the player needs and less of the item the player doesn’t.

# 3 Conclusion

UI is a major part of communicating game mechanics and information to the player. This includes the AI of a game. During our research, we realized that UI has the major role of showing that a game has AI. While UI doesn’t excel at showing the individual AI of an agent, world interaction being a better indicator, it excels at showing the overall AI of a game. This closely relates to the orchestrator, which controls the overall experience of a game.

Through UI, we can give AI the ability to communicate with the player to help them make better decisions and have curated experiences tailored to the player. This can be applied to almost any part of a game that a player interacts with, by tracking the player’s decisions and programming an AI to adjust accordingly. This can be applied to almost any game where the player makes preferential decisions, in order to track the player’s unique perspective towards the game and adjust the game space in order to better improve their experience. However, this also requires a large amount of data to be tracked and stored in order to affectively assess the player’s decision making process; meaning that it may not be cost effective to apply this system to every aspect of user interface and experience, and instead only apply it to certain and specific aspects of the game that would most benefit from it.

If we were to extend this project, we would focus on quantifying the relationship between AI and UI; figuring out how factors of the AI determine what UI is needed. This would first begin with categorizing AI and UI components and determining some quantifiable relationship. With measurable variables, we would be able to begin working on trying to apply AI on UI generation. In short, the next step to this project would be to attempt AI generated UI, or in other words, giving AI the ability to automatically communicate to the player through UI.

# 4 References

[AI is the New UI [3]] AI is the New UI. 2017. AI is About to Become Your Company’s Digital Spokesperson. Accenture. https://www.accenture.com/us-en/insight-artificial-intelligence-ui (accessed December 6, 2018).