**Erin Grover**

**GAP 321 – Module 08**

**Mall Utility-based AI Technical Documentation**

The fundamental details needed in order to gain a better understanding of the information detailed in this technical document answer the following questions: What engine will the team be using to accomplish this task? And, How will the Utility scoring system work?

To answer the former, the team will be using the Unreal Engine 4. The reason for this is due to the familiarity that the engineers have with the engine as apposed to most other options. This will result in the lowest possible development time by removing the need for the engineers to become familiar with any new development environments. Due to the bonus development time, the project will result in a higher quality end product.

To answer the remaining question, the Utility scoring system will work by, first, separating each trait that the agent has a desire to fulfill into a category that can be weighed separately among other categories. The score will be based solely off of a utility score graph (detailed below). From the agent’s perspective, this “bucketing” system [Graham] will answer the question of “Which desire do I wish to fulfill most?”.

Second, each trait that needs to be fulfilled will contain a list of all actions that potentially fulfill that trait. The actions will then be scored individually. This will allow a single action to receive a utility score separate to the categories score, and the usefulness of this step will be given in an example later in this proposal.

The remainder of this proposal will go into detail on the Utility scoring system, including the specific implementations of the designers requested functionality.

**Trait Utility Scores**

**Population**

Figure 1: A default population-utility curve.

Equation 1: Population-utility formula.

The team proposes that the overall crowdedness of the mall, or an area of, will be modeled with a quadratic curve, shown in Figure 1. The default population curve has been modeled here to be nearly linear, but a quadratic curve was chosen so that when tuned, the curve can represent an agent that might be more averse to crowded areas, or having a higher utility score at lower values.

Equation 1 depicts the equation used in Figure 1, the Utility Score, *U*, is the ratio of the *current population* to *max population*, with the tunable value, *s,* being able to change the sharpness of the curve. Any value of s lower than 1 and greater than 0 will result in a rotated quadratic, higher than 1 will result in a regular quadratic, and 1 will be linear. A rotated quadratic more easily represents an agent that gets frustrated in crowded areas, where a linear might be a default agent value, and a quadratic might represent an agent that isn’t bothered by crowds. The value of *s* should remain between 0 and 2.

**Energy**

Figure 2: A default energy-utility curve.

Equation 2: Energy-utility formula.

The energy level of an agent will be modeled with a piecewise combination of a reversed quadratic curve and a linear curve, shown in Figure 2. This was chosen in order to model the way that some team members personally felt while shopping at a mall: First they were excited to shop and walk around, then suddenly after a time had passed, they realized how tired they were. After that “crash”, they began to get tired at a more noticeable rate.

Equation 2 is the most complex of the equations within this technical document and is the only equation that does not tune very easily from a designer’s perspective. The first section of the piecewise is a simple linear curve, a ratio between the agent’s current energy to its maximum energy, however, the second section of the piecewise details an inversed quadratic curve, which does not tune well. A series of linear curves may be better used and tuned by a designer instead, so that an agent that tires more or less quickly may be modeled.

**Shop Desire**

Figure 3: A default shop desire-utility curve.

Equation 3: Shop desire-utility formula.

An agent’s desire to shop will be modeled from a simple linear curve, as the desire to shop increases, the likely-hood of an agent choosing to do so also increases, this is shown with both Figure 3 and Equation 3.

**Hunger**

Figure 4: A default hunger-utility curve.

Equation 4: Hunger-utility formula.

An agent’s hunger will be modeled by a logistics curve, so that an agent would remain satiated for a while, gain hunger very rapidly, then remain hungry at a moderate level, until slowly declining to a very hungry state. This is shown in Figure 4. In Equation 4, there are two values that are used to tune this model, *h* and *k*. The sharpness of the curve is tuned with k, and is a difficult value to tune properly, and must remain between 0 and 1.65 in order to maintain this curve’s relative shape, so it is best left as a constant. However, *h* is much more easily tuned by a designer. It represents the point at which the utility score for hunger will be at a value of 0.5, so a more gluttonous agent might have a lower *h* value than an agent that eats less often.

**Frustration**

Figure 5: A frustration-utility curve of an agent that might be less prone to fits of rage.

Equation 5: Frustration-Utility formula.

The team has decided that frustration should be a separate trait from Population. This is because there are multiple reasons that a typical mall that would frustrate a shopper, from dirty bathrooms, or waiting in line, to rude employees. Making this a separate trait will greatly increase the scalability of this system for the designers.

As shown in Figure 5, Frustration will be modeled with a logistics curve, similarly to hunger, but reversed. This will allow an agent to remain in a non-frustrated state, before rapidly shifting to a more frustrated state. In Equation 5, the values of h and k will behave similarly to those of Equation 4, where k is a constant and h will tune the point where the utility score for frustration will be at a value of 0.5.

**Action Utility Scores**

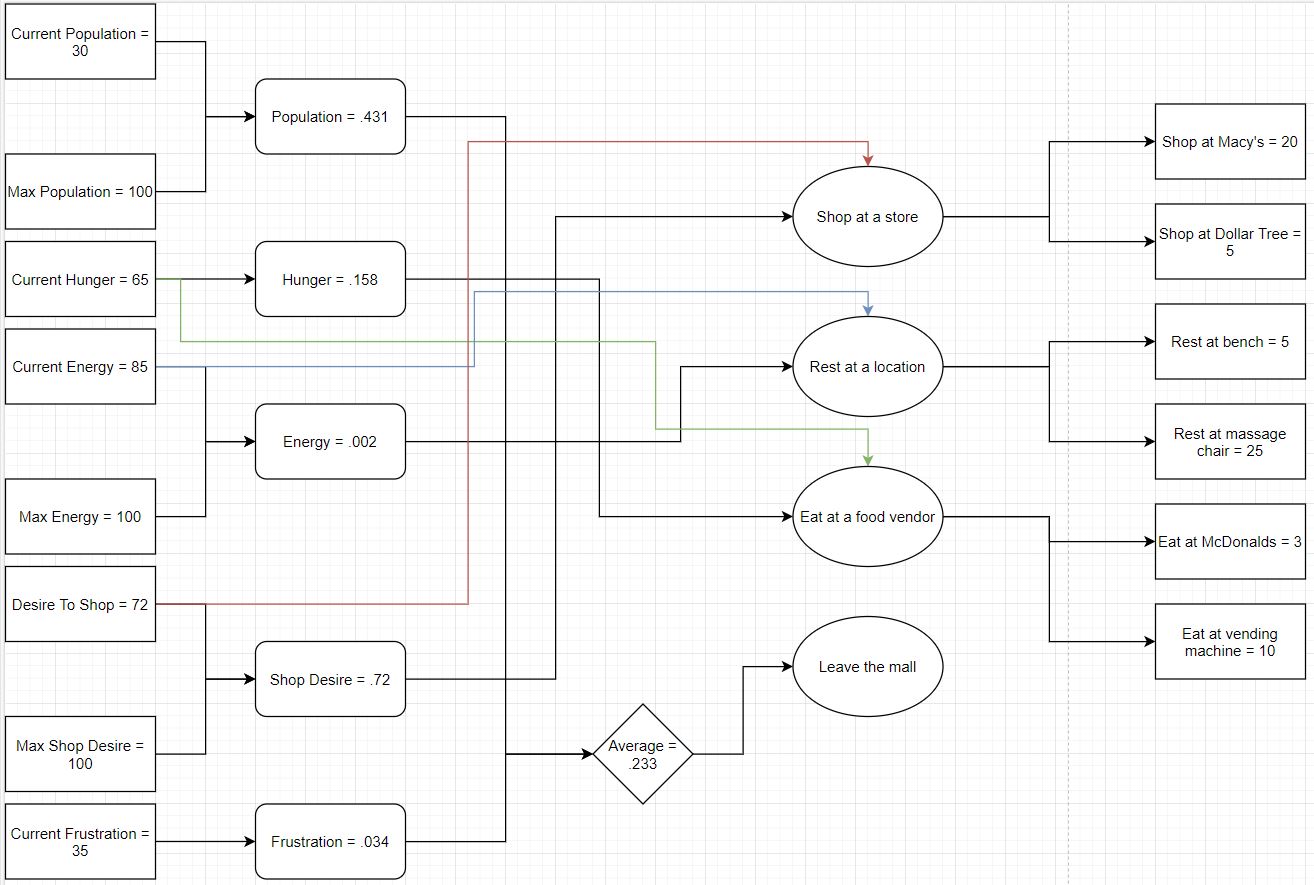


Figure 6: Example action choice diagram

Due to the designer’s request being that of a simple simulation, how an action can be scored will be made simple and designer friendly: A score will be given to an action based on how well it will fulfill the trait it belongs to.

For example, if a bench can rest an agent by 5, and a massage chair can rest an agent my 25, the agent will likely choose the massage chair. However, in order to prevent the massage chair from always getting chosen, we will compare the potential energy gain, to the energy the agent needs. Therefore, if the agent only needs 5 energy, then the massage chair would also only have a score of 5, making the two options equally comparable.

Additionally, an action will be chosen by random from among the top designer tunable amount of the highest scoring actions. To prevent an agent from oscillating between choices, an agent will not be allowed to make another choice until its current action is completed. In Figure 6, it is shown using the values taken from the equations in this document what choice would be made by the agent. If the randomization is ruled out, it is likely that the agent will choose to shop at Macy’s.

Lastly, additional factors can be added to the final score for each action. For example, resting at a massage chair can have a cost of 5 dollars, where a bench may be free. If an agent’s current wallet contains $5, they may not be willing to choose that option over a free one. This would necessitate an additional utility scoring model for an agent’s money.

**Performance**

The agents will have to iterate through each trait on every AI frame. The number of agents, a, have a number of traits, n, which also act as a container for actions, m. However, only one trait will have to check each of its actions, unless they cannot be completed, in which case, the agent should choose from the next highest scored trait. So, worst-case would be O(a\*n\*m) if all traits had an equal number of actions. This would be fine for smaller values of a, n, or m, but it does not scale well. A solution would be to not continue to iterate through the traits and actions, but instead to increase an agent’s frustration, and wait until the next AI frame to re-attempt the action selection. This would instead result in every agent scoring every trait, but only scoring actions from one trait.

**Memory**

The memory usage of this system would be negligible.