# GitHub

nshaffer26 ([nshaffer26/GameAI (github.com)](https://github.com/nshaffer26/GameAI))

# Class Diagram

Graphical user interface, diagram

Description automatically generated

# Reasoning

I decided to implement Goal Oriented Action Planning (GOAP) because it was the architecture I felt was the most interesting. Additionally, I thought it would fit reasonably well with my simulation idea, which was a restaurant containing customers and waiters. That being said, I did run into some issues regarding how cost fits into things and also how preconditions are determined. Overall, I think I made a good choice using GOAP, but in hindsight I think a Blackboard system could have also worked because it enables all agents to have knowledge of the global world state, not just their own. This might have created a bit more flexibility and maybe even variety in my simulation, because as it stands, the behavior chain of each individual agent is relatively simple and straightforward, and their plan is almost, if not exactly, the same with each iteration.

# Design

My AI architecture consists of two different types of agents with multiple individual actions each. I made use of Goal Oriented Action Planning (GOAP) to create a restaurant where customers sit down, wait to be brought food by a waiter, eat, and then leave.

I also make use of a small Finite State Machine (FSM) similar to the one shown in class for the game *F.E.A.R.* The states in this FSM are as follows:

* **IDLE:** If an agent is in this state, they are forming a plan
* **GOTO:** If an agent is in this state, they are moving to an action so they can perform that action
* **ACTION:** If an agent is in this state, they are attempting to perform an action

When in the IDLE state, each agent will examine the world state and create a plan of actions in the hopes of satisfying their goal(s). A tree is created by examining all valid actions for an agent (i.e., all actions which have their preconditions met based on the world state), with each node in the tree representing a different action, and leaves representing actions which meet the goal state. Nodes are chained together by examining the world state and finding actions whose preconditions are being met. If they are met, a new node is created for this action and the action’s effects are applied to the world state temporarily (the planning phase does not actually perform the actions). Now that the world state has been updated, the remaining valid actions are examined to see if another node can be created. Since each action has a cost, each node also keeps track of the running cost for that branch up until that node. A plan is formulated from this tree by examining all branches and determining which leaf node has the lowest running cost. This branch then becomes the plan, and the agent moves into the ACTION state (or GOTO if they are not in range of the target for that action) starting with the first action in the branch. The agent will continue to transition between the ACTION and GOTO states until all actions in the plan are complete, at which point they will transition to the IDLE state and formulate another plan.

Each agent’s goals and actions are detailed below, along with each action’s cost, preconditions, and effects.

## Outline

### Customer

#### Goals

* **sitDown**
* **eatFood**

#### Actions

* SitDown
  + Cost: 1.0
  + Preconditions: ~sittingDown
  + Effect: sittingDown, sitDown
  + Other Conditions: There is an empty seat available.
* Eat
  + Cost: 1.0
  + Preconditions: isHungry
  + Effect: ~isHungry
  + Other Conditions: There is food nearby.
* Leave
  + Cost: 1.0
  + Preconditions: ~isHungry
  + Effect: ~sittingDown, **eatFood**

### Waiter

#### Goals

* **collectFood**
* **severCustomer**

#### Actions

* CollectFood
  + Cost: 1.0
  + Preconditions: canHoldMoreFood
  + Effect: hasFood, **collectFood**
  + Other Conditions: There is food at the counter and the counter is nearby.
* DeliverFood
  + Cost: 1.0 or -1.0
  + Preconditions: hasFood
  + Effect: canHoldMoreFood, **serveCustomer**
  + Other Conditions: There is a customer nearby who is waiting for food. If this agent is closer to a waiting customer than to the food counter, the cost for this action is -1.0. Otherwise, it is 1.0.

# Questions

I’m not sure if any of these have already been answered, but some of my questions are as follows

* I was a bit confused about the difference between the GotoNode action of *F.E.A.R.*’s GOAP architecture and the Goto state their FSM. Would you want to choose between one or the other, or is it best to have both?
* I also experienced some confusion regarding how to effectively implement costs. For example, my waiter agents can only hold two items of food at once. Before implementing cost, the waiters would form the same plan of action every time, that is, CollectFood > DeliverFood. This was a bit problematic and looked weird because a waiter would go back to the counter and collect food every time they delivered some, even if they were already holding food and did not need to collect more. To solve this, I added a collectFood goal to the waiter agent and gave deliver food a cost of -1 if they are closer to a waiting customer than they are to the food counter (and a cost of 1 otherwise). Although this implementation created the behavior I was looking for, I’m not completely certain if it was the best way or even right way in terms of GOAP, as it started to feel like I was manufacturing situations rather than letting the AI decide.