# Goal Oriented Action Planning AI

PERSONAL PROGRAMMING PROJECT

CHECK-IN PRESENTATION

BY JORDAN MARTIN

# Objectives

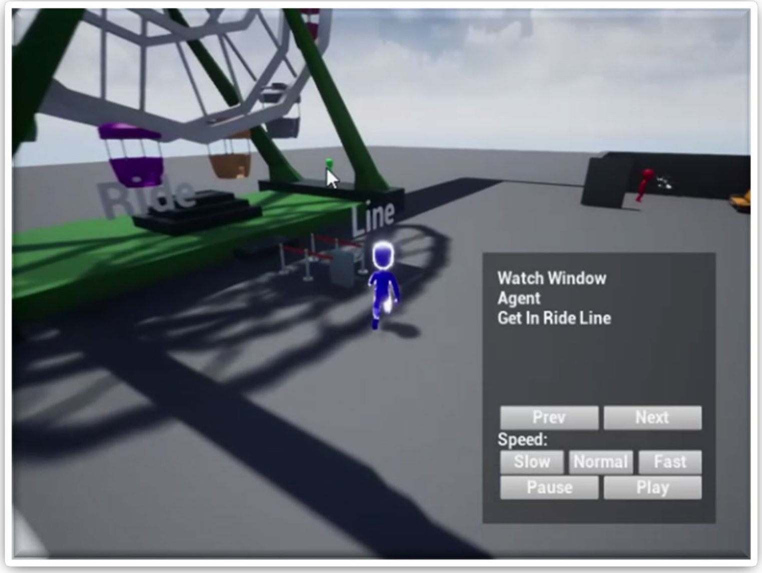
- Create a GOAP AI system in **Unreal**
- Integrate the GOAP AI system into an agent-based simulation
- Create a theme park scenario demonstration

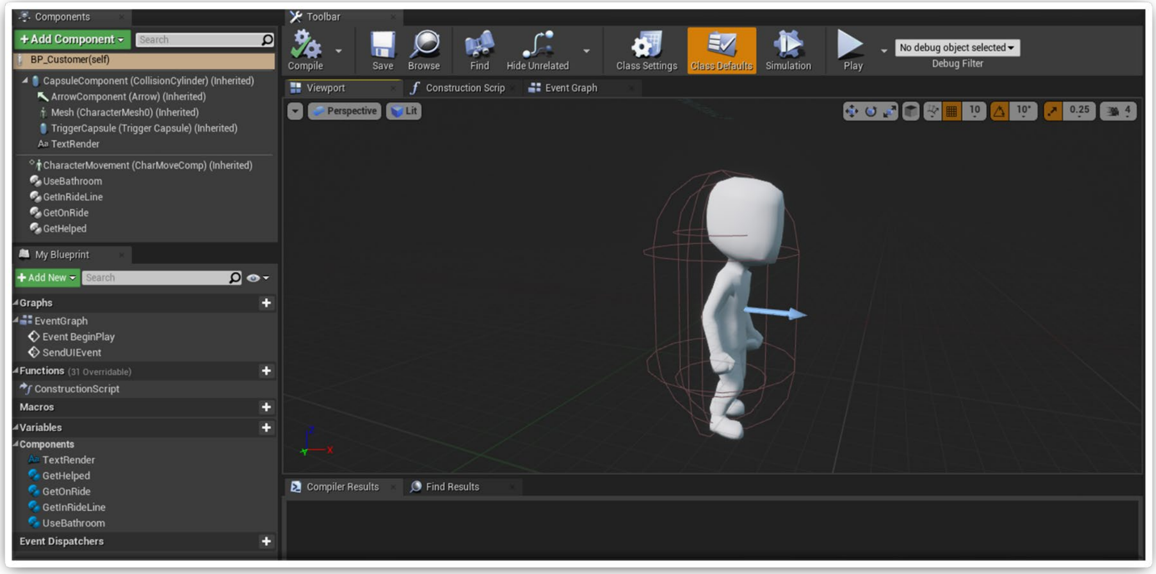
Stretch Goals:

- Incorporate game mechanics into the simulation

# Progress

- GOAP System created in C++ classes (no behavior trees)
- Multi-step plans and multiple agents
- Inventories
- Game elements: RTS Camera, UI Watch Window, Speed
- Unreal C++: Collision, Spawning, Data Structures, Delegates


Watch Window
Agent
Get In Ride Line



# GOAP System

Made of 5 C++ Classes:

**Agent** - Any of the NPCs that will be controlled using the GOAP system

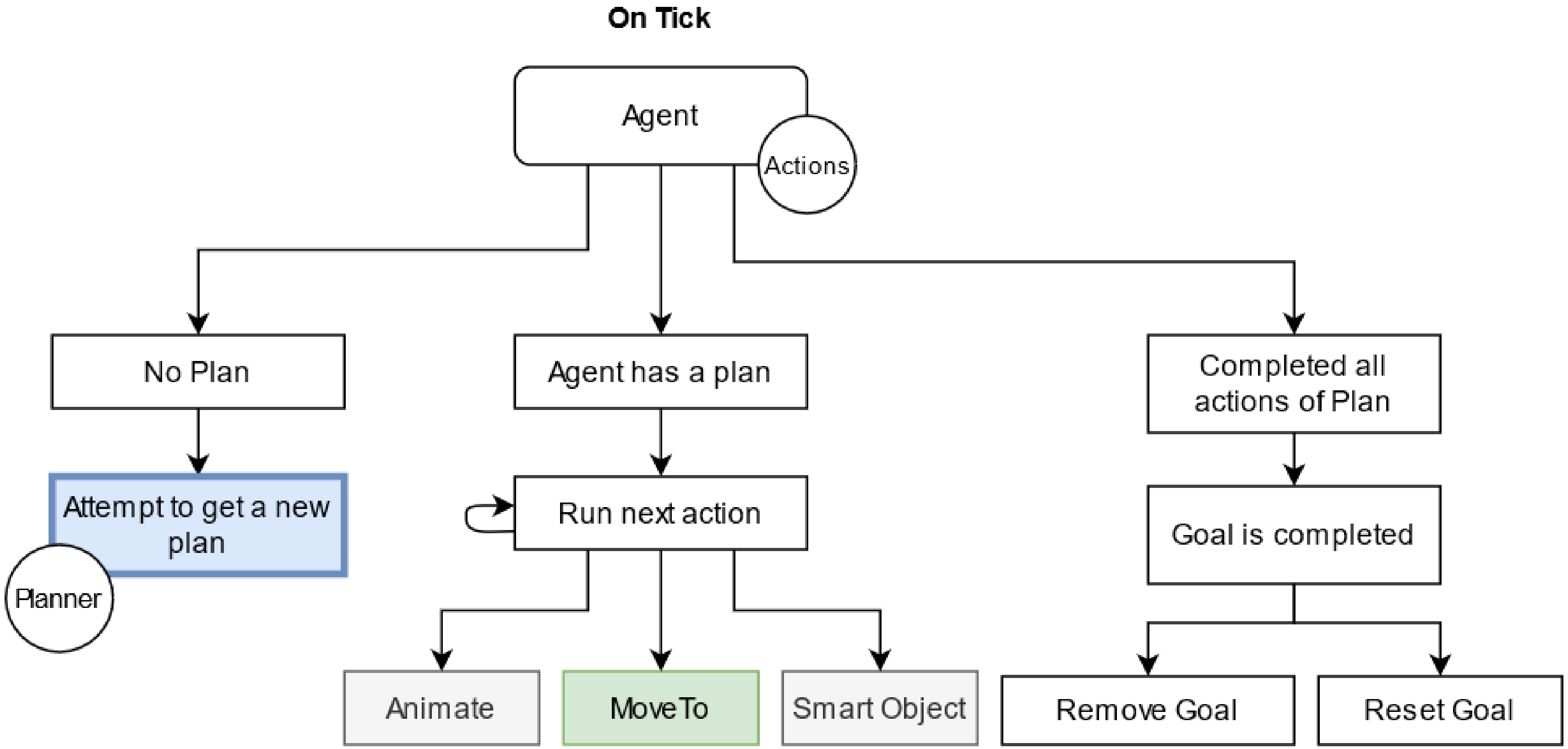
**Action** - Any of the actions that an agent use to achieve a goal

**Planner** - Builds a graph of possible plans and sorts it based on cost of actions

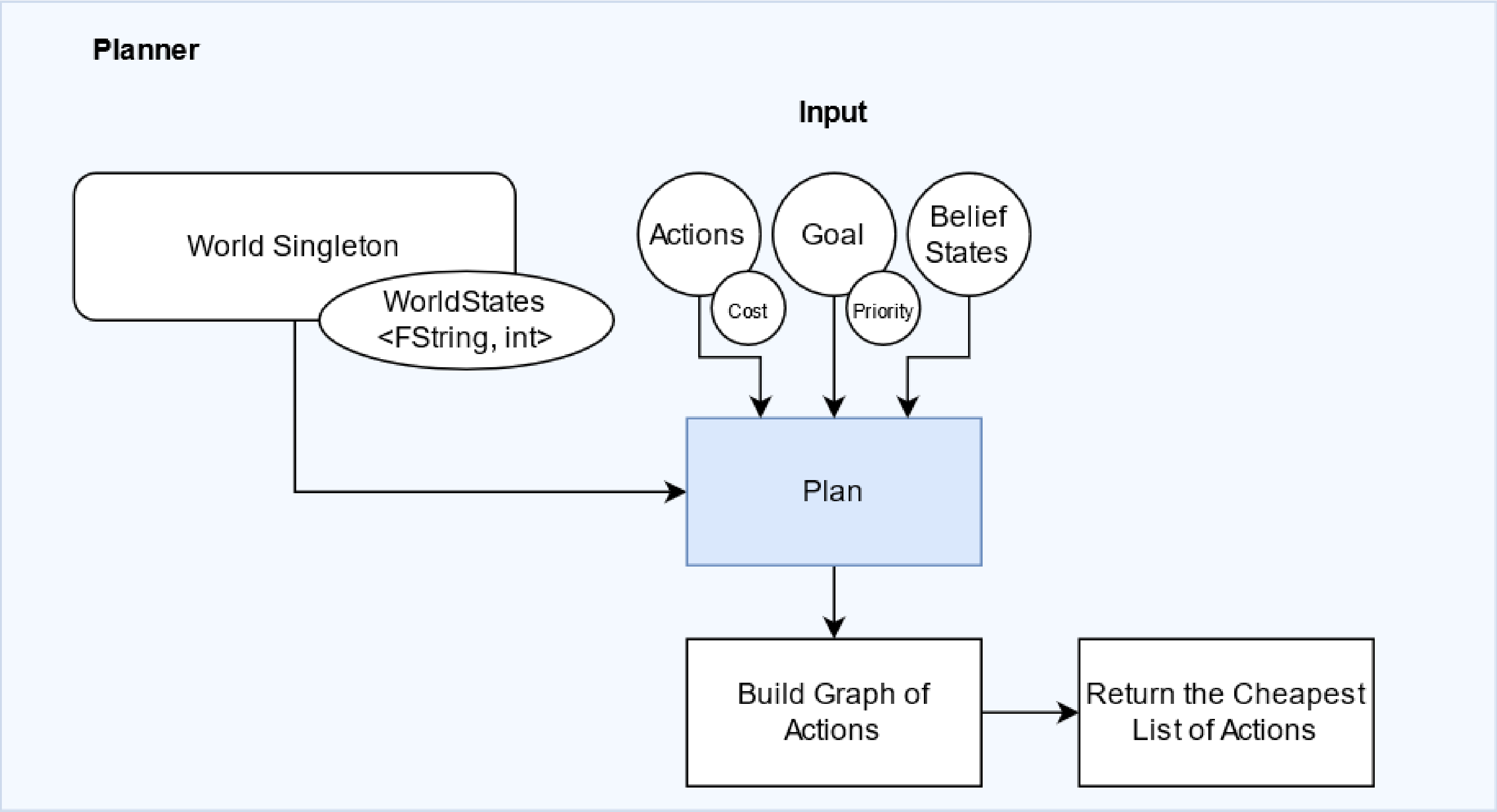
**World States** - A class that maintains a map of pairs representing states of the world

**World** - A singleton that allows access to the map of WorldStates

# GOAP System: Agent

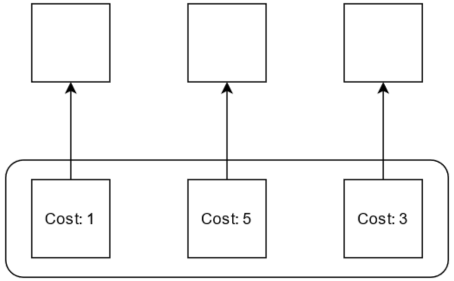


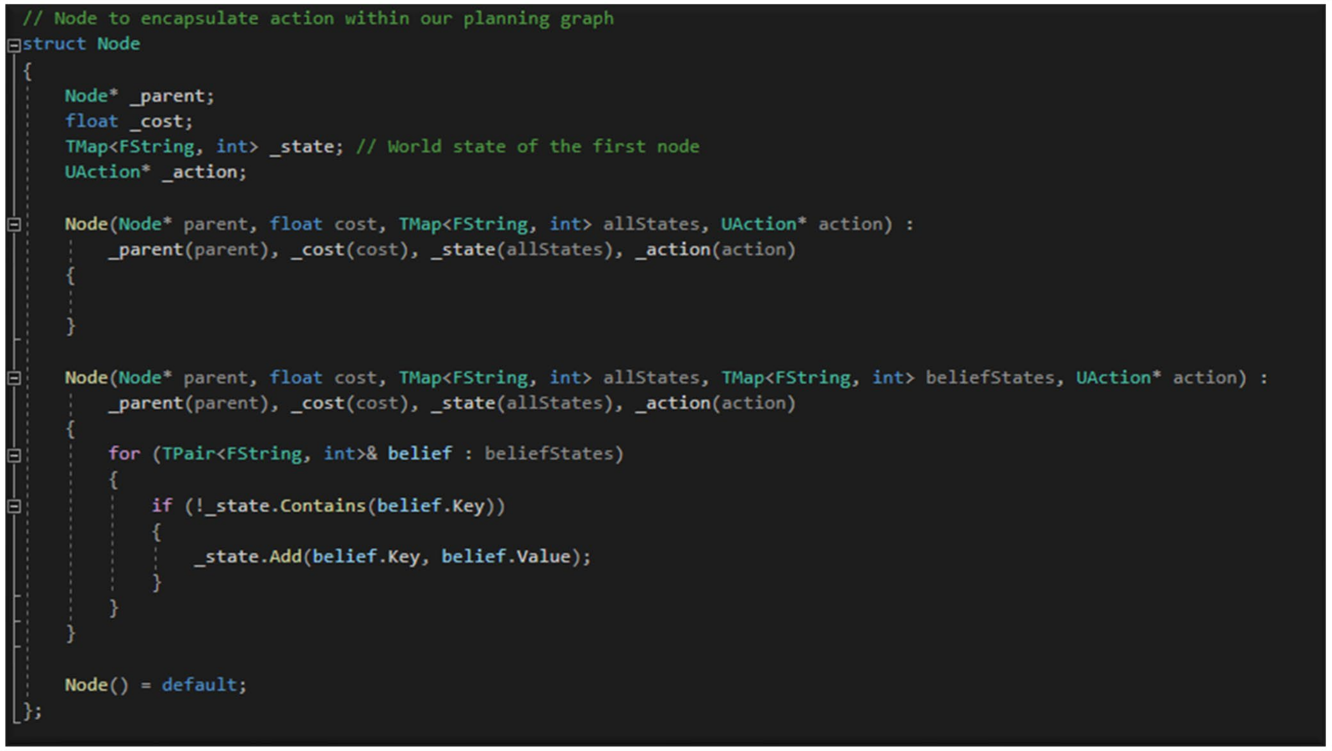
# GOAP System: Plan



# Planner

- Uses a linked list style node to build graphs of actions
- Possible plans are returned as a vector of linked lists and the total cost of each list is compared to find the cheapest plan

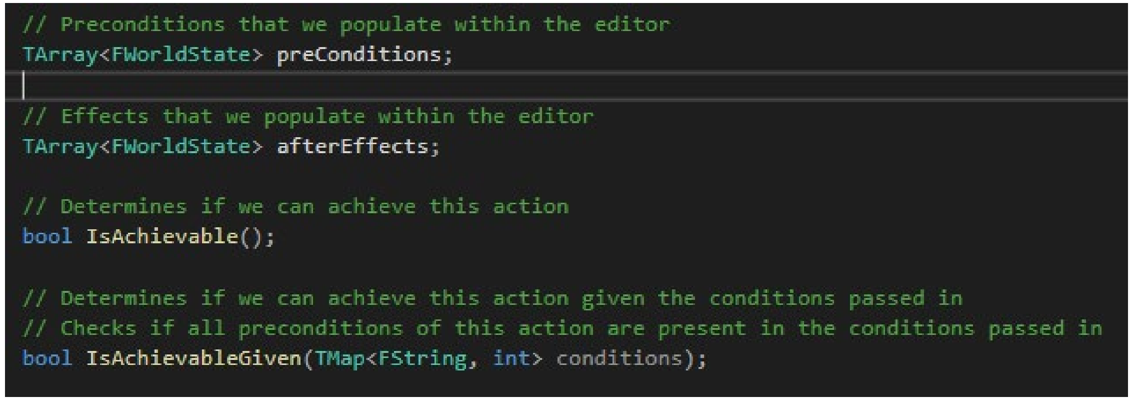




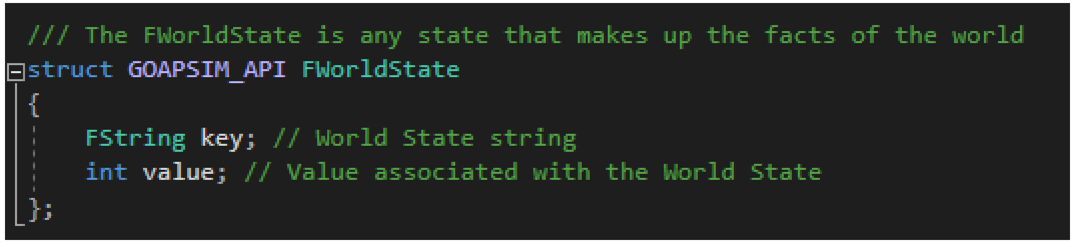
```
// Node to encapsulate action within our planning graph
struct Node
{
    Node* _parent;
    float _cost;
    TMap<FString, int> _state; // World state of the first node
    UAction* _action;

    Node(Node* parent, float cost, TMap<FString, int> allStates, UAction* action) :
        _parent(parent), _cost(cost), _state(allStates), _action(action)
    {

    }

    Node(Node* parent, float cost, TMap<FString, int> allStates, TMap<FString, int> beliefStates, UAction* action) :
        _parent(parent), _cost(cost), _state(allStates), _action(action)
    {
        for (TPair<FString, int>& belief : beliefStates)
        {
            if (!_state.Contains(belief.Key))
            {
                _state.Add(belief.Key, belief.Value);
            }
        }
    }

    Node() = default;
};
```

# Action

- Contains a list of preconditions and after-effects to be checked with the WorldStates
- Determines if the action is achievable given the current WorldState



```
// Preconditions that we populate within the editor
TArray<FWorldState> preConditions;

// Effects that we populate within the editor
TArray<FWorldState> afterEffects;

// Determines if we can achieve this action
bool IsAchievable();

// Determines if we can achieve this action given the conditions passed in
// Checks if all preconditions of this action are present in the conditions passed in
bool IsAchievableGiven(TMap<FString, int> conditions);
```



```
/// The FWorldState is any state that makes up the facts of the world
struct GOAPSIM_API FWorldState
{
    FString key; // World State string
    int value; // Value associated with the World State
};
```

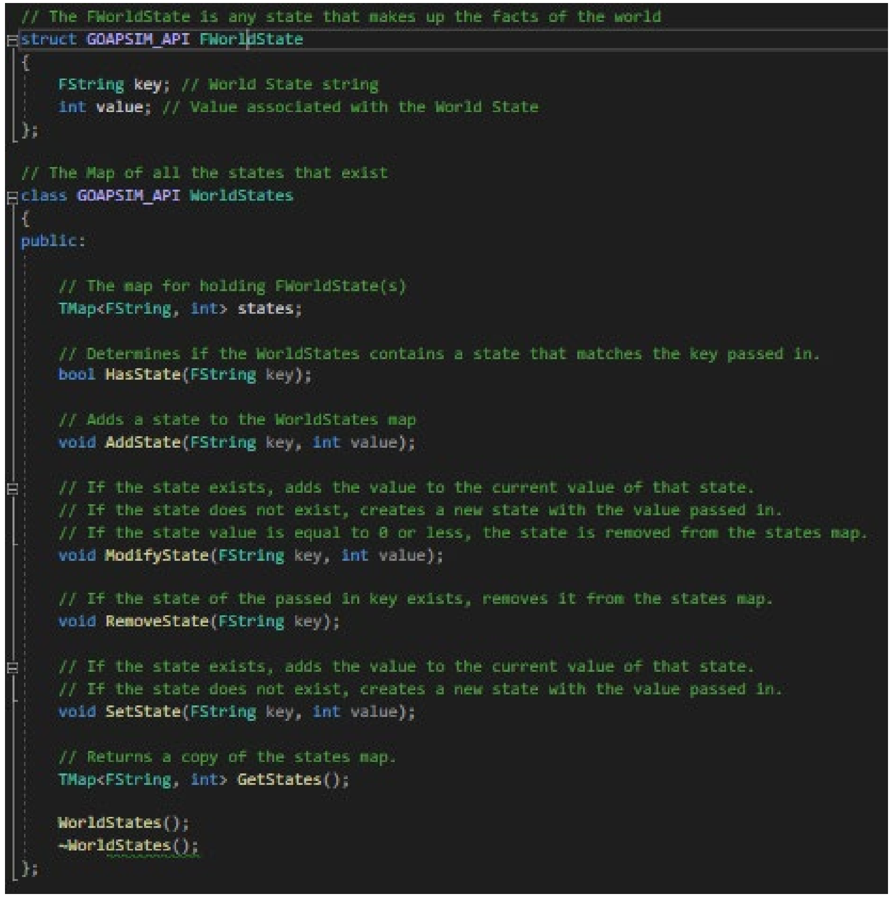
# World and World States

World States

- Each state represents a fact within the world
- Contains methods to check, modify, add, and remove states

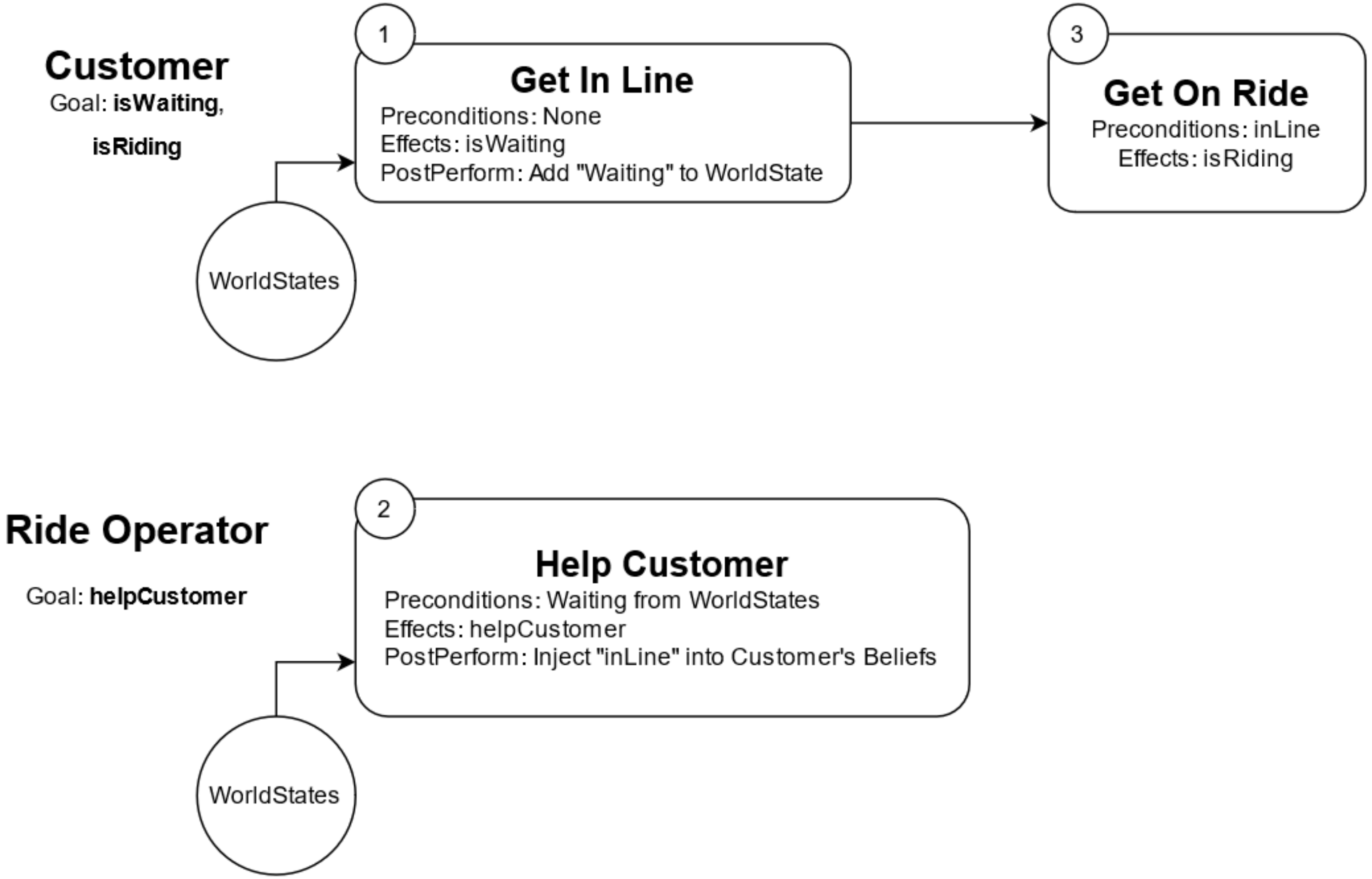
World Singleton

- Allow access to the WorldStates



```cpp
// The FWorldState is any state that makes up the facts of the world
struct GOAPSIM_API FWorldState
{
    FString key; // World State string
    int value; // Value associated with the World State
};

// The Map of all the states that exist
class GOAPSIM_API WorldStates
{
public:

    // The map for holding FWorldState(s)
    TMap<FString, int> states;

    // Determines if the WorldStates contains a state that matches the key passed in.
    bool HasState(FString key);

    // Adds a state to the WorldStates map
    void AddState(FString key, int value);

    // If the state exists, adds the value to the current value of that state.
    // If the state does not exist, creates a new state with the value passed in.
    // If the state value is equal to 0 or less, the state is removed from the states map.
    void ModifyState(FString key, int value);

    // If the state of the passed in key exists, removes it from the states map.
    void RemoveState(FString key);

    // If the state exists, adds the value to the current value of that state.
    // If the state does not exist, creates a new state with the value passed in.
    void SetState(FString key, int value);

    // Returns a copy of the states map.
    TMap<FString, int> GetStates();

    WorldStates();
    ~WorldStates();
};
```

# Example



**Customer**

Goal: **isWaiting**, **isRiding**

1 **Get In Line**

Preconditions: None

Effects: isWaiting

PostPerform: Add "Waiting" to WorldState

3 **Get On Ride**

Preconditions: inLine

Effects: isRiding

WorldStates

**Ride Operator**

Goal: **helpCustomer**

2 **Help Customer**

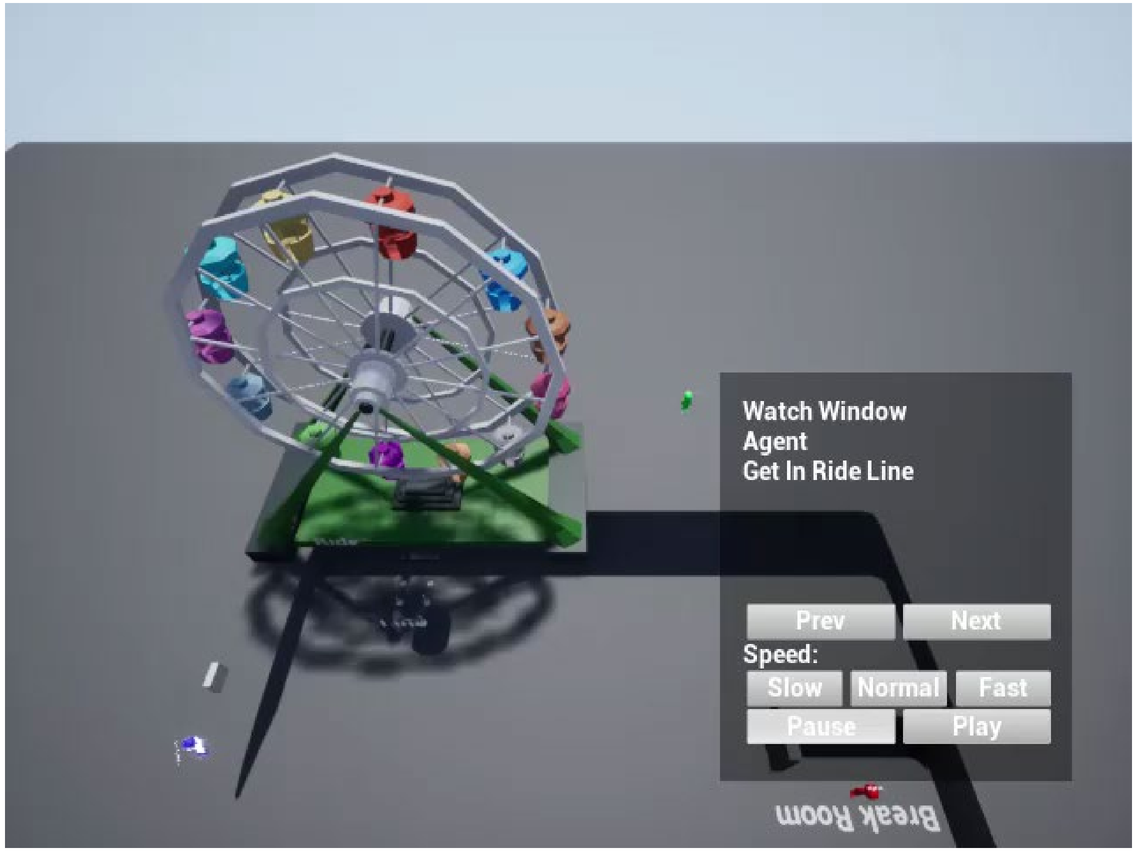
Preconditions: Waiting from WorldStates

Effects: helpCustomer

PostPerform: Inject "inLine" into Customer's Beliefs

WorldStates

# Demo Video



Watch Window
Agent
Get In Ride Line

# Previous Schedule

| | | |
|---|---|---|
| **Creating the GOAP AI System** | Week 1 | **Proposal Presentation**<br>• Further Research<br>• Agent simulation moving with Basic NavMeshes |
| | Week 2 | Creating the Environment<br>• World State, Actions, Agents Classes<br>• Spawning of agents |
| | Week 3 | Creating the Planner<br>• Executing a simple plan on an agent |
| | Week 4 | Expanding on the Planner<br>• Executing multi-step plans on multiple agents |
| | Week 5 | Monitoring Agents State in Real Time<br>• Debug tool to give details through UI<br>• Inventory system<br>• Speed Buttons<br>• RTS Camera |

🟦 Additional

# Future Schedule

| Goal | Week | Previous Schedule | New Schedule Revisions |
|---|---|---|---|
| **Integrating GOAP System into Agent-Based Simulation** | Week 6 | **Update Presentation**<br>Adding “Smart Objects” to the world<br>• Add objects that can be used to fulfill goals | **Update Presentation**<br>• Complete multi-step plans<br>• Flesh out UI Window<br>• Animate State |
| | Week 7 | Revalidation of plans<br>• Add changes to the world state that require plan changes | Adding “Smart Objects” to the world<br>• Add objects that can be used to fulfill goals |
| | Week 8 | Priority of Goals for agents<br>• Execute plans while having competing priorities | Revalidation of plans<br>• Add changes to the world state that require plan changes |
| **Applications towards Games and Simulation** | Week 9 | Create a tycoon game<br>• Add player agency to the game and have planner adjust | Priority of Goals for agents<br>• Execute plans while having competing priorities |
| | Week 10 | Create a tycoon game<br>• UI, Resources, Placement of Buildings, Game Logic | Create a Theme Park Simulation<br>• Implement all agents and actions<br>• Integrate GOAP system into Demo Scene<br>• Time for fixes and polish |
| | Week 11 | **Final Presentations** | |

🟦 Additional

# *Thank you!*