Applying GOAP to Tic Tac Toe and Checkers

By LeeKingRyan

Finite State Machine (FSM)

A Finite State Machine is a model of computation based on a hypothetical machine based on one or more states. Only a single state can be active at the same time, thus the machine must transition from one state to another to perform different actions.

For graph representation, the nodes are the states and the edges are the transitions with each edge having a label that informs when the transition occurs.

In the case of games, character actions can be broken into states/nodes and are linked through transitions based on conditions.

What is Goal-Oriented Action Programing (GOAP)

The GOAP architecture is a decision-making architecture, essentially a planner that formulates a sequence of actions to satisfy some goal. An agent is given such plans.

A goal is a condition an agent wants to satisfy, and since a sequence of actions are enacted, then a goal control's a character's behavior. Additionally, a goal can calculate its relevance, and knows when it's been satisfied. For example, IsValid(). A goal simply defines a set of conditions needed to be met, not an embedded plan.

A plan is a sequence of actions, and a plan is ONLY valid if it take a character from a start state to some state that satisfies the CURRENT goal (goals can be invalidated), thus another IsValid() method is used for plans.

An action is a single step that makes a character act. Actions know when it's valid to run, so an action has ***preconditions*** and ***effects***. The preconditions and effects are the essential components to chain actions into a valid sequence, as some actions require their preconditions to be met first. Meanwhile, an actions effects changes the world state and may be the preconditions of another action.

Applying GOAP to Tic Tac Toe

Generate a plan by supply goals to a planner, then the planner searches the space of actions for a sequence that takes the character from the starting state to goal state. Note: Each Goal and Action has an IsValid() check function.

Goals	Desired World_State (conditions)
Win (final move)	bool won: true
Prevent_Win	bool blockPlayer: true (route is given externally)
Check_Mate (Boards for a guaranteed win)	bool boardSet: true (specific to board)

Actions	Preconditions	Effects
Mark()	—	(area on board is marked)
DrawLine()	(have three consecutive marks including diagonal)	bool won: true

The Planner and A* search

Within the planner are nodes which are representations of the state of the game world, and the edges are the actions. The A* algorithm will calculate the cost of a node and the heuristic distance from a node to a goal. “The cost of a node can be calculated as the sum of the costs of the actions that take the world to the state represented by the node”.

Action costs may vary, while heuristic distance “can be calculated as the sum of the unsatisfied properties of the goal state”. Searching will be done regressively.

Winning Boards in Heuristic Order

	X	
X		X

		X
	X	
		X

X		X
	X	

X		
	X	
X		

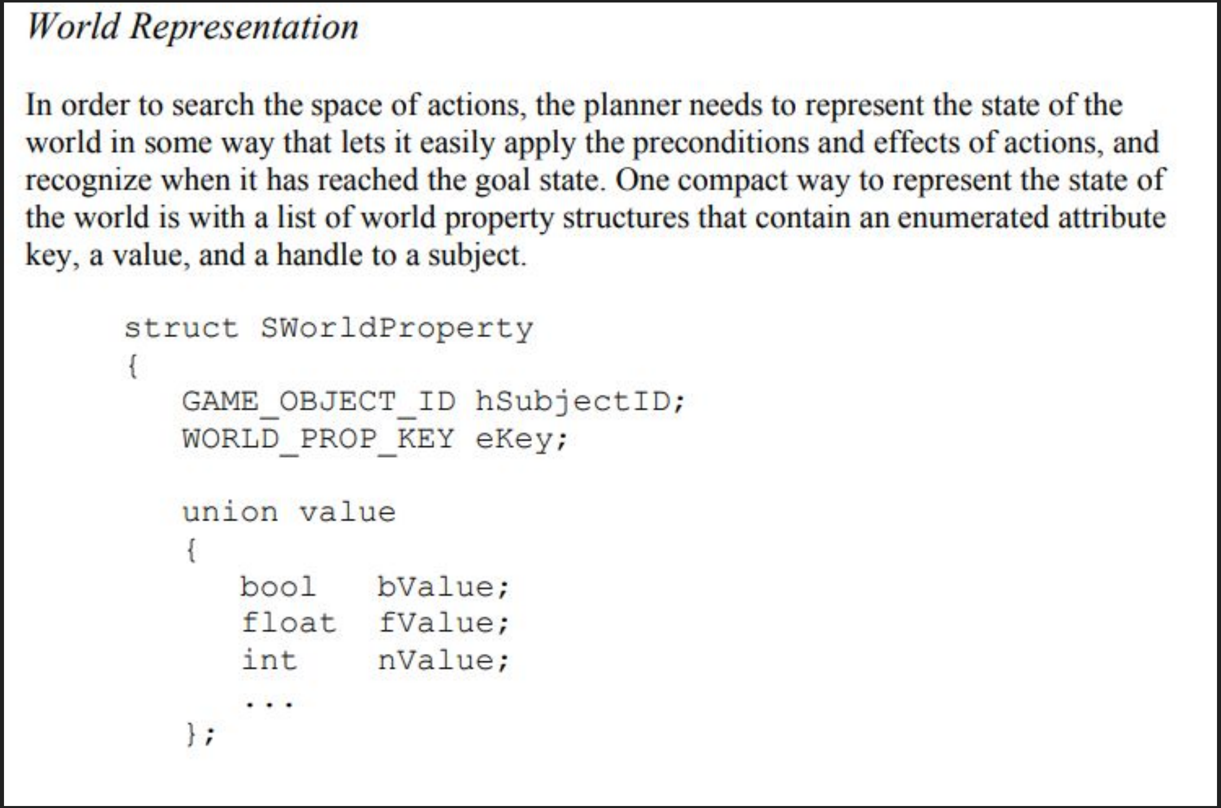
X		X
		X

		X
X		X

X		
X		X

X		X
X		

World Representation Structure



World Representation

In order to search the space of actions, the planner needs to represent the state of the world in some way that lets it easily apply the preconditions and effects of actions, and recognize when it has reached the goal state. One compact way to represent the state of the world is with a list of world property structures that contain an enumerated attribute key, a value, and a handle to a subject.

```
struct SWorldProperty
{
   GAME_OBJECT_ID hSubjectID;
   WORLD_PROP_KEY eKey;

   union value
   {
      bool   bValue;
      float  fValue;
      int    nValue;
      ...
   };
```

Formulating a Plan with A* Search

GOAP and FSM

The FSM in F.E.A.R. is composed of only three states. GoTo already has prescription animations associated to walking and running. Simply the characters navigate to a location, and can transition to the animate state to perform some animation. Use Smart Object just plays an animation assigned to an object separate from characters.

With GOAP, when a plan is executed, the actions are sequentially activated, “which in turn set the current state, and any associated parameters”.

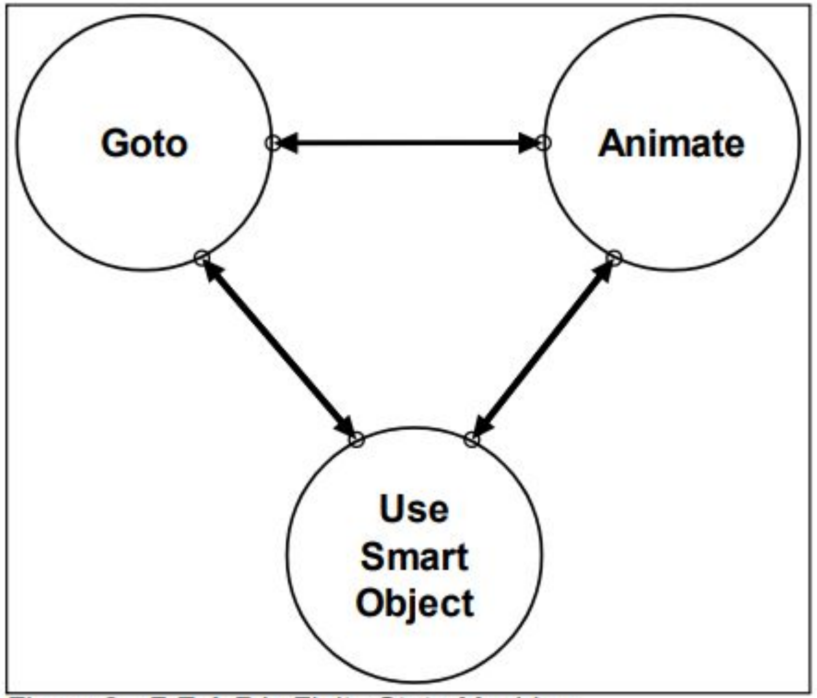


Figure 2: *F.E.A.R.*’s Finite State Machine

Summary for Tic Tac TOe

- Low level subsystem decides who plays first
- The current World State is set
- Player or AI goes first:
- GOAP architecture (planner) chooses a valid goal through the hierarchy
- AI creates a plan to accomplish current goal based of current World State
- If a valid plan is chosen, the FSM system executes it, until completion or if the plan later becomes invalid. More specifically, an action in the plan can adjust the parameters for a specific state. For example, Mark() makes a location or index in an array of bools set to true, for GoTo() to move a piece and Animate() will animate the associated animation. (UseSmartObject may not be a necessary state for this project).
- Plan become invalid, then a new goal is searched, then a new plan is made accordingly
- Low level subsystem checks whether game is over or not.

Applying GOAP to Checkers

Why GOAP & Future Plans

GOAP allows more convenient addition to character actions and goals rather than coding individual behavior trees for characters and scripted events. With the planning system, our small team can toss in goals and actions, so we’ll only have to worry about design of environment and characters and behavior.

The same GOAP architecture can be implemented onto other development projects without conflictions.

Debugging process is made easier, as actions and goals can be decoupled and tested individually.

Dynamic Problem solving in real-time by AI, hence more diverse actions in games appropriate to player action and world status. Give the illusion of intelligent AI.

Resources about GOAP

https://alumni.media.mit.edu/~jorkin/goap.html

https://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf

https://alumni.media.mit.edu/~jorkin/GOAP_draft_AIWisdom2_2003.pdf

https://www.youtube.com/watch?v=gm7K68663rA&t=1377s&pp=ygUdR29hbCBvcmllbnRlZCBhY3Rpb24gcGxhbm5pbmc%3D

https://www.youtube.com/watch?v=LhnlNKWh7oc&pp=ygUdR29hbCBvcmllbnRlZCBhY3Rpb24gcGxhbm5pbmc%3D

https://www.youtube.com/watch?v=nEnNtiumgIl&pp=ygUdR29hbCBvcmllbnRlZCBhY3Rpb24gcGxhbm5pbmc%3D

https://www.youtube.com/watch?v=qaIF0iaDWRQ&pp=ygUdR29hbCBvcmllbnRlZCBhY3Rpb24gcGxhbm5pbmc%3D

AI Tutorials and Playlist including GOAP and FSM in Unity

https://www.youtube.com/watch?v=qaIF0iaDWRQ&list=PLpgZM2kb9mwtnzD9qTg8VkAaFIOkUM_j8&pp=iAQB

https://youtube.com/watch?v=n6vn7d5R_2c&feature=share

https://www.youtube.com/watch?v=cwlFbLLR3qc&list=PLkBiJgxNbuOXBAN5aJnMVkQ9yRSB1UYrG&pp=iAQB

Other Resources including CodeMonkey Tutorials

https://www.youtube.com/watch?v=cwlFbLLR3qc&list=PLkBiJgxNbuOXBAN5aJnMVkQ9yRSB1UYrG&pp=iAQB