Video Games and AI Planning

Artificial Intelligence @ Allegheny College

Janyl Jumadinova

October 16, 2018
AI in (commercial) video games
AI in (commercial) video games

- Path finding
- Realistic motion
- Models of emotion
- Decision making
- Learning
- Nonlinear story telling
- ...

Janyl Jumadinova
AI techniques are currently used for the decision making of non-player characters (NPCs) in the commercial video games.
Finite State Machines

NPC behavior based on high-level states.
Finite State Machines

- Traditionally one of the first techniques for NPC behavior.
- Very simple to understand.
- Very simple to implement (e.g., every state is a method).
Behavior Trees

NPC behavior based on more refined conditions and strategies.

Diagram:
- Door open?
- Move into room

→
NPC behavior based on more refined conditions and strategies.

Unity Behavior Tree
NPC behavior based on more refined conditions and strategies (e.g. Halo 2).
Goal Oriented Action Planning (GOAP)

Search in real-time for a strategy that achieves the goal in the current state (F.E.A.R).
Goal Oriented Action Planning (GOAP)

Each action is associated with a cost. GOAP takes actions' preconditions and effects and creates a list of actions that will reach the goal.

FSM

GOAP
Goal Oriented Action Planning (GOAP)

- Each action is associated with a cost.
- GOAP takes actions’ preconditions and effects and creates a list of actions that will reach the goal.

GOAP tutorial
iThink: STRIPS planning in Unity3D

- [https://code.google.com/p/ithink-unity3d/](https://code.google.com/p/ithink-unity3d/)
- Provides a basic framework for specifying and solving STRIPS planning problems inside Unity3D.
- Can be used as a realistic testbed to try STRIPS planning in commercial games.
Gaming

- Game Design (How are games created?)
- Game Genres
Gaming

- Game Design (How are games created?)
- Game Genres (first-person shooter, racing, strategy)
- Game Editor
- Game Element (players, goals, narrative, decisions, balance)
- Game Play
Gaming

- Game Design (How are games created?)
- Game Genres (first-person shooter, racing, strategy)
- Game Editor
- Game Element (players, goals, narrative, decisions, balance)
- Game Play (rules, interactive modes, challenge types, level design)
Gaming Industry

- Americans spent a record $36 billion on video games in 2017. That is $10 billion more on content in 2017 than they did in 2012 (a 54% increase).
- The number is expected to go up in 2018.
Customer preferred gaming platforms according to gaming companies worldwide as of August 2016

- Mobile: 76%
- PC/Mac: 62%
- Social/online: 40%
- Console: 29%
Unity

Simplify development of games

- Game Mechanics
  - physics, AI
- Rendering Effects
  - shadows, lighting
- I/O Abstraction
  - input devices
  - output devices
Unity Learning Interface Tutorial Editor Basics

Janyl Jumadinova

Video Games and AI Planning

October 16, 2018
Unity

Assets/NewBehaviourScript.cs(14,9): error CS1525: Unexpected symbol `}`

Assets/NewBehaviourScript.cs(16,1): error CS8025: Parsing error

Assets/NewBehaviourScript.cs(14,9): error CS1525: Unexpected symbol `}`
Examples: https://unity.com/madewith

Documentation:
https://docs.unity3d.com/Manual/UnityOverview.html
Unity
Asset:
representation of any item that can be used in your game or project.
Game Objects

- Every object in the Unity game is a **GameObject** - characters, lights, cameras, and special effects.
- **GameObject** can’t do anything on its own
  - have to give it properties before it can become a character, an environment, or a special effect
  - give properties by adding **components** to it
  - use Unity Scripting API
void Start()
{
    //called once. do any initialization here
}

void Update()
{
    //called every frame. move objects here
}
Sample Game

https://unity3d.com/learn/tutorials/s/roll-ball-tutorial
Extending the Sample Game

**Make Items**

- Create a new Game Object (any shape), name it “Item”.
- Tag the item as “item” by selecting Tags, and creating a new tag called “item”.
- Now go back to Tags for that game object and select the new “item” tag that you created.
- Tag all your items as items. Make sure your spelling matches exactly.
- Place the Item into an empty Game Object called “Items”.
- Reset their transforms.
- Add a rigidbody to the Item.
- Duplicate the Item a few times and place them around the arena.
Extending the Sample Game

Player to Collect the Items and Display the Score

- Open the player movement script from the Inspector panel with the Player game object selected.
- Make two declarations for (1) a variable that keeps track of the score and (2) a GUI text that displays the score on the scene view.

```csharp
private int count;
public GUIText countText;
```

- Under the function `void Start()`, initialize `count` and `CountText`

```csharp
count=0;
CountText();
```
Extending the Sample Game

cont. Player to Collect the Items and Display the Score

- Write a new function for collision with the Items.
  ```csharp
  void OnTriggerEnter(Collider other) {
    if (other.gameObject.tag == "item") {
      other.gameObject.SetActive(false);
      count=count+1;
      CountText();
    }
  }
  ```

- Write the CountText function, which will update the score on the GUI display.
  ```csharp
  Void CountText() {
    countText.text="Count: " + count.ToString();
  }
  ```
Extending the Sample Game

cont. Player to Collect the Items and Display the Score

- Save the program and go back to Unity.
- Select all your items, make sure they are tagged as items, and check the button “Is Trigger” in the Box Collider component of the Inspector.
- Check the “Is Kinematic” button under rigidbody to prevent your items from falling through the floor.
- For the `countText`, create a new GUI Text using the Create option under Hierarchy.
- Set the GUI Text’s transform to (0,1,0) and give it a pixel offset of (10, -10) in the GUIText component on the Inspector panel.
- Drag the GUI Text into the Count Text box on the Inspector with the Player selected.
Extending the Sample Game

**Make Hazards**

- Create new empty game object called “Hazards”.
- Create a new Quad and call it “Hazard”. Tag it as hazard, and check “Is Trigger”.
- Change its color by selecting Mesh Renderer in the Inspector, with the hazard selected, and changing its material.
- Click the drop-down by Materials, and use the little gray circle to the right of the box to select a different material than the default gray one for the hazard.
- Change the hazard’s rotation to 90 about the X axis and lower its Y height to -0.4 so it is a small white square lying just over the floor of the arena.
Edit the Player script, under the `OnTriggerEnter()` function. What is its purpose?

```csharp
void OnTriggerEnter(Collider other) {
    if(other.gameObject.tag=="item") {
        other.gameObject.SetActive(false);
        count = count + 1;
        CountText();
    }
    if(other.gameObject.tag=="hazard") {
        other.gameObject.SetActive(false);
        Vector3 jump = new Vector3(0.0f, 30, 0.0f);
        rigidbody.AddForce (jump * speed * Time.deltaTime);
    }
}
```
cont. Make Hazards

- Save the program, go back to the Unity editor, and duplicate the hazard a few times.
- Position the hazards around the arena, and try out the game!