Artificial Intelligence for Games: Basics
Thursday, Mar 31, 2016

Reading: Some of the material from today’s lecture comes from the book “Artificial Intelligence for Games” (2nd Edition) by I. Millington and J. Funge.

What is Artificial Intelligence? Artificial intelligence (AI) can be defined (circularly) as “the study of computational systems that exhibit intelligence.” Unfortunately, it is not easy to define what we mean by “intelligence.” In the context of games, an in particular in the design of non-player characters (NPCs) a working definition might be, “It is whatever a person would do.” (Where, of course, the word “person” might be replaced by “soldier,” “zombie,” or “enchanted unicorn,” whatever makes sense for the current context.)

At a basic level, game entities have goals that they are expected to achieve (e.g., staying out of danger, pursuing the enemy, fighting). This leads to a view of AI as planning strategies to achieve these goals. Computing optimal ways of achieving these goals may involve optimization algorithms at a low level (e.g., find the shortest path from here to there) progressing up to complex planning strategies at a high level (e.g., assemble a bunch of wooden crates in order to form a stable structure making it possible to climb out of a pit).

Often in games, AI is most evident in games when it fails, that is, when nonplaying characters behave in an inexplicably nonsensical manner. (For example, a pedestrian character that continues to walk nonchalantly down the street in the midst of a gun fight.)

Roles of Game AI: Generally, AI is used in games is to determine complex behaviors that not specified by the player nor a direct effect of physics. Examples include:

Nonplayer Opponents: In a first-person shooter game, opponents should exhibit realistic attack behavior, which might include a decreased level or aggression or even retreating when suffering damage.

Nonplayer Teammates: Given a squadron of soldiers, the group should move in a coordinated supportive manner. Such support NPCs are sometimes employed in multiplayer online games to assist inexperienced players. While in some contexts, this might be scripted by the game designer, typically this is handled by the game’s AI system.

Support and Autonomous Characters: This includes generating realistic crowd behavior, where the characters may need to interact in a realistic manner when coming into contact with the player’s character.

Commentary/Instruction: Again, this is typically scripted, but an example requiring AI might involve determining whether the player is stuck and in need of a hint on how to proceed.

The key element in all of these examples is the feature of complexity. Examples of things that are not AI include:

Determined by physical laws: Examples include the way in which a basketball bounces off the rim of a basket, or the spinning motion of a car that just hit an obstacle.
**Purely random:** For example, the shape of the next block that falls in a game of Tetris.

**Direct response to game rules/user inputs:** This includes events for which the response is predetermined by the game designer. This includes typical camera control, scripted animations, events that are triggered by the user’s inputs, and events that are scheduled to occur at a particular time or after a particular time delay.

One notable gray area is where AI ends and animation begins. For example, a soccer player dribbling the ball must make decisions as to how to avoid opponents, which in turn affects the direction and speed with which he/she runs, which in turn affects joint angles. Typically, AI systems control the high-level decisions and the animation controls the lower level decisions:

**Should I run with the ball or pass it?** Definitely an AI decision (unless scripted or determined by user input).

**If I run, what path should I take?** This is getting into the gray area. If we wish to evaluate the likelihood of success of various options, based on hypotheses of how the player and other NPCs might respond, we are in the realm of AI. If we simply wish to compute a shortest obstacle-avoiding path (say using Dijkstra’s algorithm), this is in the realm of **algorithmics**.​[1]

**How to move my legs to travel along this path?** Now we are definitely outside of the realm of AI and into the realm of animation.

**Properties of a Good AI System:** The following is a list of generally good properties of a game AI system.

**Goal driven:** The AI system should behave in a manner that is consistent with the (implicit) high-level goals of the entities involved.

**Responsive:** The AI system should respond rapidly to relevant changes in the state of the world. For example, if a path is blocked, the NPC should respond quickly by computing a new path.

**Smart, but not omniscient:** The AI system should behave as if it knows a good deal about the world (inanimate objects, other NPCs, and even the player) and select its behaviors accordingly. Of course, an NPC cannot act based on information that it could not reasonably have knowledge of (such as the position of the player, if the player is not close enough to be seen or heard).

**Consistent:** An NPC should behave in a consistent manner, to generate the impression that it embodies a believable character.

**Efficient and Practical:** Computational resources are limited, and the time needed to develop, program, and test the AI system must be considered within the economic constraints of the game.

Unfortunately, many of these goals conflict with each other, and many of the problems in game AI result from developers making compromises in quality for the sake of simplicity or efficiency.

---

[1] Some might claim that this is AI, since algorithmics is just an offshoot of AI. If you say this in earshot of an algorithms researcher, be prepared to get punched in the face.
The AI Loop: The fundamental view of a game from the perspective of AI involves continuously evaluating the current state of the (perceptible) world and determining what should the agent do in the very near future (that is, the next frame to be drawn). This generally may involve a number of layers of sensing and decision making. These are illustrated in Fig. 1.

![Game AI Loop Diagram]

Fig. 1: The game AI loop.

Perception: The entity senses the elements of the game state within its scope of perception.

Model update and outcome: The perception of the state results in updates to the entity's internal state model. The outcome of these state changes may be very simple (“I am injured and need to retreat”) but could conceivably be quite sophisticated in the context of a game with complex narrative structure (“An ally has acted against my interests. Is he/she a spy?”)

Goals and intentions: Based on a character’s understanding of the world state, what are its motivations, and how are these motivations weighed to arrive at a set of goals? These goals need to be mapped into intentions that are to be manifest through the character’s future actions.

Plan and action: Given these intentions, the character then needs to develop a plan of actions in order to achieve the desired results. Such a plan consists of a sequence of tasks. Once a plan has been developed, the character needs to act in order to perform these tasks, and may need to update the plan if conditions change.

These concepts are not perfectly cut and dry, but as we proceed from goals to intentions to plans to actions, we are decreasing the scope of each decision and increasing the specificity of the decisions being made.

AI Execution Management: AI can account for a considerable amount of the processing time in a game. Because of its complex nature, much of the AI processing is done within the CPU (as opposed to the GPU, where most graphics processing is performed). The AI system receives information from the game system about the state of the world. This may occur by querying the state of the world or through some sort of message-passing system, where actions elicited by one game object (“Thrust spear”) are transmitted to other game objects (“Inform me whenever I am attacked”). This information must be processed both by individual agents to make the necessary decisions to determine their actions as well as to groups of agents, who employ some strategy to coordinate their actions. These actions are then transmitted...
to other components of the game system (such as animation and physics) to carry out these actions (see Fig. 2).

Fig. 2: AI Execution Management. (Source: Millington and Funge)

**Scheduling:** In a complex game, there may be many game objects that are competing for the same computational resources to solve their individual AI tasks. For example, you might have a large number of non-playing characters. Some are fighting against the player, while others are just milling around in the background. Clearly, the characters involved in the fight have the highest priority for the computational resources, because they need to respond quickly to the player's actions, and poor or unnatural performance will be spotted immediately by the player. For background characters, once an AI task has been solved (e.g., the path to follow to walk down a corridor), we may not need to perform this task again for some time in the future. Some AI processing might involve lengthy computations, but because of the need for interactivity, the various tasks will need to be broken up into small time slices to be executed periodically (e.g., once every tenth of a second).

An example of the sort of task scheduling for a small fragment of a game is shown in Fig. 3. Think of each update cycle as a single call to the Unity function FixedUpdate (meaning that it is invoked at regular intervals, normally once every tenth of a second).

Scheduling tasks is an intriguing computational problem. Basically, you are simulating a multi-threaded programming environment, but for the sake of efficiency and control, you may prefer to do your own scheduling rather than letting the operating system (or programming language run-time system) do it for you. First, tasks need to be broken up into small executable fragments. Next, these fragments need to be scheduled (assigned) to update cycles, so that:

(i) Each fragment is executed frequently enough to maintain real-time performance for this AI entity.
Fig. 3: Example of AI task scheduling. (Source: Millington and Funge)

(ii) Each update cycle completes quickly enough so that the game’s frame-rate is not adversely affected.

Millington and Funge observe that the simplest way to achieve objective (i) is to schedule each fragment at regular intervals, depending on the frequency with which it needs to be executed. However, since frequencies vary between task fragments, there may be **clumping**, where an excessive number of subtasks are scheduled to be executed at the same time. An example is shown in Fig. 4 for three subtasks (a, b, and c) which have respective frequencies of being executed every 2, 3, and 4 update cycles, respectively.

Fig. 4: The problem of clumping when regular periodic scheduling is used. (Source: Millington and Funge)

Millington and Funge suggest a number of strategies\(^2\) for producing schedules that satisfy both requirements (i) and (ii).

**Agents:** NPCs are often modeled in games through the use of an AI construct called an **agent**, which is defined to be an entity situated within and a part of an environment that senses

---

\(^2\)But frankly, I found none of them interesting enough to report here.
that environment and acts (over time) in pursuit of its own goals and so as to effect what it senses in the future.

At a high conceptual level, an agent is characterized by three basic components, which operate iteratively over time:

**Sensing**: Perceive features of the environment, and in particular, changes to the environment that are relevant to the agent’s goals.

**Thinking**: Decide what action to take to achieve its goals, given the current situation and its knowledge. Of course, this is where all the complexity lies. Thinking may be simple reaction (“Danger. Flee!”) or may involve complex responses based on past experiences and learning. If the objectives are complex, apply planning strategies to break them into well-defined actions.

**Acting**: Carry out these actions.

**A Taxonomy of Agents**: Agents come in many forms and with many degrees of sophistication. Here is a general classification of agents based on the complexity of their decision-making processing.

**Simple reflex agent**: Such an agent reacts solely based on its current perception of the world, without regard to prior experiences or its current state. These are usually implemented by simple rule-based systems. “If $X$ occurs then do $Y$.“ (E.g., “If the player appears, I attack.”) This can be encoded as a table, where input events (mapped to integer indices) are looked-up in a table of actions.

**Model-based agent**: This extends the simple reflex agent by storing its own internal state (“I’m healthy/hungry/injured.”). Such an agent determines an action based on both the state of the world and its own internal state. These agents are often implemented by a finite-state machine. The machine’s state corresponds to the agent’s state, and current perception triggers an action and a transition to a possibly different state (For example, “If I am wandering and healthy (state) and see the player’s avatar (current perception), I will start pursuing it (action).”)

**Goal-based agent**: This type of agent further extends on the capabilities of the model-based agents, by using goal information. Goal information describes longer-term states that are desirable and our actions will (eventually) lead to achieving these goals. This allows the agent a way to select among multiple possible intentions and selecting one (ideally the best one) in order to achieve the goal state. Search and planning algorithms may be invoked to map goals into low-level actions.

**Utility-based agent**: This is a further extension of the goal-based agent by defining a measure of how desirable a particular state is. This measure is expressed through the use of a utility function, which measures how happy the agent would be with this state. The agent then chooses the action that maximizes the expected utility of the action (see Fig.5).

**Learning agent**: Such an agent takes the utility-based approach one step further by evaluating the results of past actions, and then uses this information to make (hopefully) better
choices in the future. ("Last time I hid here I was found, so now I'll hide somewhere else.")

It has two important components, a learning element, which is responsible for making improvements by critically evaluating the benefit of the outcomes of prior actions, and performance element, which is responsible for selecting external actions. Note that future actions may not be chosen solely on the basis of expected utility, but there may also be exploration of unknown states to determine their utility.

Fig. 5: Structure of a utility-based agent. (Source: Millington and Funge)

Looking ahead: In future lectures, we will explore two important aspects of AI in computer games. The first is planning motion to achieve various goals (e.g., pursue the enemy by the shortest path) subject to various constraints (e.g., avoid obstacles). We will also discuss techniques for making decisions.