# GADE Ice Task 1 (class task)

**What is AI in gaming?**

AI can be used for any things in gaming. AI is usually used to generate a game environment or content and even to control characters within the game. AI can also be used to manage systems or game mechanics (ie: calculating a players score based on actions). AI is often used to optimise and/or perform processes to make the gaming experience better for the players. For example, AI can be used to create characters within the game that players can interact with, play with and play against.

**How is knowledge used in gaming?**

Many games make use of knowledge by encouraging the player to acquire information. As the player learns more about the game or progresses further, the player will become more competent. The game’s AI can then learn from the behaviour of the players, becoming more optimised as more players learn to play the game.

**Activity 1.1.1 (on learn):**

1. AI techniques are methods such as machine learning, deep learning, natural language processing, and computer vision that AI can use to learn and adapt.
2. League of Legends is a 5v5 MOBA game that has an AI game mode. Here the player can play alongside up to 4 AI bots against a team of up to 5 AI bots. The game uses AI to automate the bots - they need to make decisions based on weapon loadouts, attacking enemies, and overall game strategies. This can be considered real AI as the bots are making their own decisions based on what is happening in the game.
3. These methods are considered real AI because each bot will behave differently. The bots will not have the same reaction every time they are presented with a scenario, they will behave differently depending on the game situation.

**A 3-state FSM pseudocode program:**

A finite state machine diagram for a single move 2 player game (like chess)


**Pseudocode:**

Class process

{

Class StateTransition

{

State activeState;

Command command;

Public StateTransition (activeState, command)

{

activeState = activeState;

Command = command;

}

}

Dictionary <StateTransition, State> transitions;

Public State activeState {get; set;}

Public Process()

{

activeState = State.idle;

Transitions = new Dictionary <StateTransition, State>

{

// add a state transition for each transition in the state machine

// maximum of 4

};

}

Public State StateNext (command)

{

new StateTransition{activeState, command);

State = nextState;

If (NOT transitions.TryGetValue(transition, out nextState))

Throw exception

return nextState;

}

Public State MoveNext (command)

{

activeState = StateNext(command);

return activeState;

}

}

Class Program

{

Static void Main(string[] args)

{

Process p = new Process();

Write “current state” + p.activeState;

}

}

**How the state machine works:**

The SM (state machine) uses a dictionary to store the four transitions described in the diagram (page 1). This FSM was designed for a game like *Chess* where there are very few states, in this case only three: idle, calculating and action.

The program holds the basic logic for the FSM: the methods that will change the state depending on the command given. This can be used to create an AI opponent in a game. Once the AI’s turn begins, the SM will be activated through the following commands: begin, end, and transition.

Performing moves, calculating moves and processing the opponents moves will be actions executed depending on the current state.