ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

IMPLEMENTACIONES PROPIAS
Y
TECNOLOGÍAS EMERGENTES

Por: Lucas Merenciano Martínez

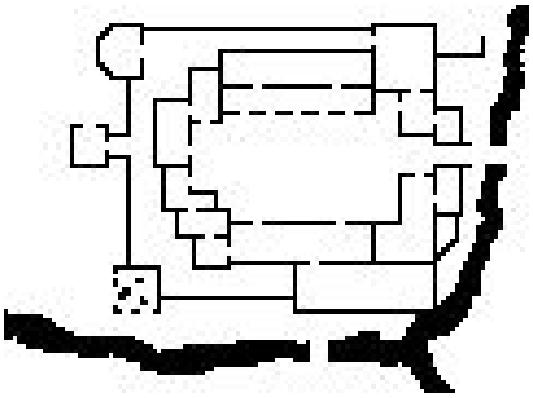
Nombre Asignatura: Inteligencia Artificial
Modulo Asociado: Unit 27: Artificial Intelligence (L5)
Profesor: Antonio Barella
Curso: 2019/2020
Autor: Lucas Merenciano Martínez

Índice

Implementaciones propias

Gestión y transformación de la información del mapa para el cálculo de caminos

Para el cálculo de caminos utilizando el algoritmo A*, la única información que necesito de el mapa son las áreas por las que los agentes se pueden mover.
Para tener esta información fácilmente, tengo preparada previamente una imagen que representa el mapa formada por colores.

1) 

2) 

1) Ejemplo de imagen que contiene información de las áreas navegables y no navegables por los agentes.

2) Esta imagen es el mapa actual que verá el usuario de la aplicación, cuya información de zonas navegables se corresponde al mapa anterior (Figura 1).

Para extraer la información, se carga el fichero en memoria, y accediendo a la coordenada deseada (correspondiente a un pixel concreto) se extrae el color de esa posición.
La forma de interpretar el color varía dependiendo de cómo esté preparada la imágen, siendo en este caso que si el píxel leído contiene el color negro, esa posición no es navegable.
En este caso con eso ya sería suficiente, pero podemos expandir los usos de estas imágenes añadiendo más colores que nos den más información, por ejemplo la velocidad a la que se podrá navegar por esa zona.

Para tener esta información más a mano y poder permitirnos descargar la imagen de memoria, una correcta implementación sería leer la imagen durante la carga del mapa e ir rellenando un array de dos dimensiones (o un bitmap) con la información extraída de la imagen. Si almacenamos esta información en la estructura que representa el mapa, estará disponible para que el algoritmo A* conozca por qué zonas puede navegar.

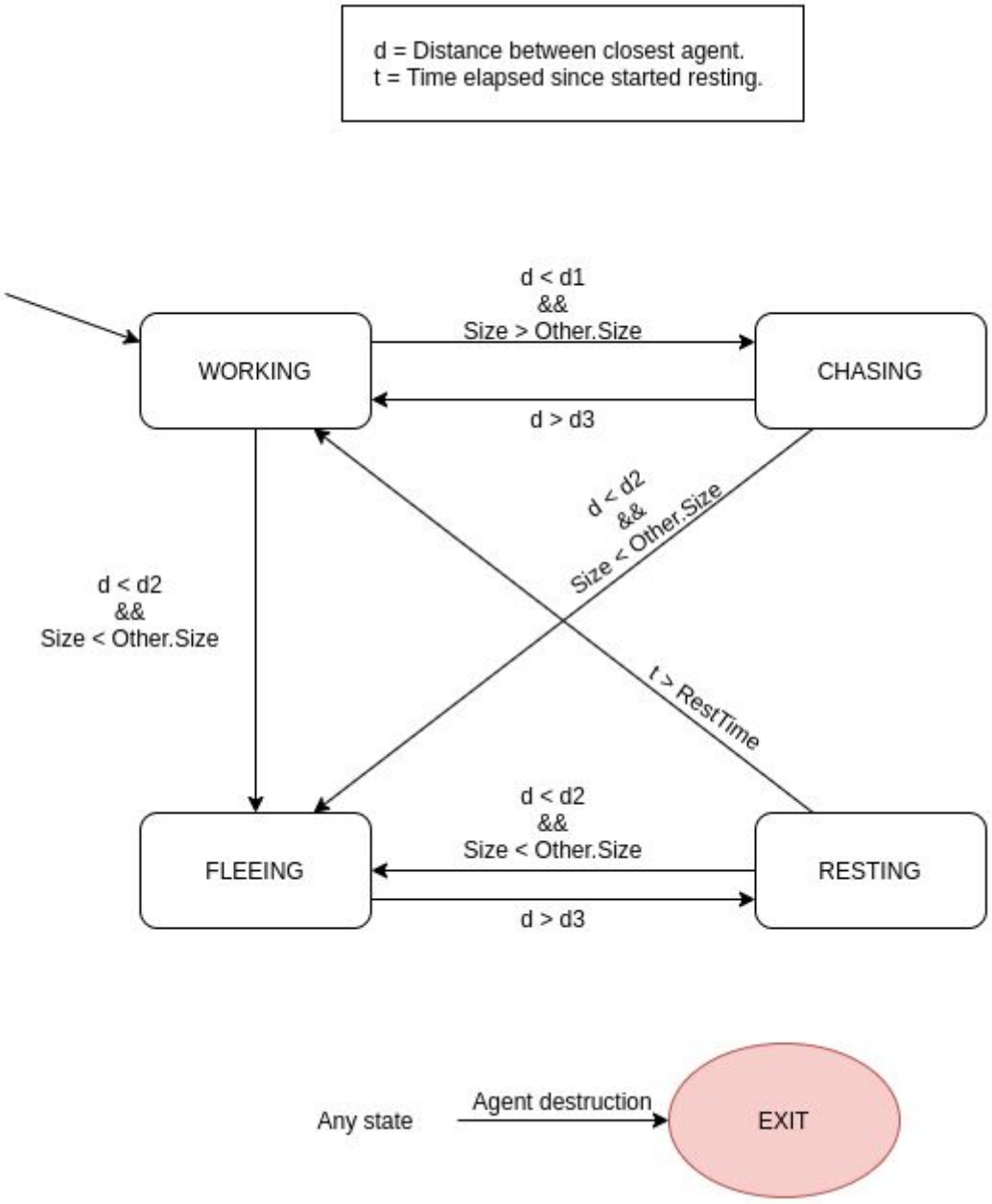
Estado y comportamiento de los agentes utilizando FSM

Se pueden utilizar muchas técnicas de inteligencia artificial para programar el comportamiento de los agentes, pero dadas las necesidades de los agentes de este ejemplo, se utiliza una máquina de estados finita (FSM).

Requisitos

- El agente tiene que incluir los estados: working, chasing, fleeing, resting.
- El estado por defecto de un agente es working. El tipo de movimiento por defecto se aplica durante el estado working.
- Un agente persigue a otro si está a una distancia menor que **d1** y su tamaño es mayor.
- Un agente huye de otro si está a una distancia menor que **d2** y su tamaño es menor.
- Un agente deja de huir o perseguir si está a una distancia mayor que **d3**, en cuyo caso pasa a descansar durante un intervalo de tiempo.
- Un agente que descansa puede huir, pero no persigue hasta que acabe de descansar.

Diseño



3)

3) Representación del diseño de la FSM a implementar.

Implementación

Para la implementación de esta máquina de estados, se definen los diferentes estados utilizando un enumerador. Y en el bucle de simulación, utilizando un switch, primero se comprueban las condiciones para cambiar de estado si se da el caso, y después se ejecuta el comportamiento de ese estado.

4)

```
enum AgentState {
  kStateNone = 0,
  kStateWorking,
  kStateChasing,
  kStateFleeing,
  kStateResting,
};
```

5)

```
void Agent::updateState()
{
  float reaction_distance = 0.0f;
  switch(size_)
  {
    case kSmall: reaction_distance = 250.0f; break;
    case kNormal: reaction_distance = 225.0f; break;
    case kHuge: reaction_distance = 200.0f; break;
    default: assert(false && "Default case sertting reaction distance."); break;
  }
  // closest_agent will be nullptr if there is no agent closer than reaction_distance
  Agent *closest_agent = CheckDistances(*this, reaction_distance);
  switch (state_)
  {
    case kStateWorking:
      if (closest_agent != nullptr)
      {
        if (closest_agent->size_ > size_)
        {
          if (type_ == MovType::kMovPattern)
            target_reached_ = false;
          state_ = AgentState::kStateFleeing;
          target_entity_ = closest_agent;
        }
        else if (closest_agent->size_ < size_)
        {
          if (type_ == MovType::kMovPattern)
            target_reached_ = false;
          state_ = AgentState::kStateChasing;
          target_entity_ = closest_agent;
        }
      }
      break;
```

6)

```
case kStateChasing:
  if (closest_agent == nullptr)
  {
    state_ = AgentState::kStateWorking;
    target_entity_ = nullptr;
  }
  else
  {
    if (closest_agent->size_ > size_)
    {
      state_ = AgentState::kStateFleeing;
      target_entity_ = closest_agent;
    }
    else if (closest_agent->size_ < size_)
    {
      target_entity_ = closest_agent;
    }
  }
  break;
```

7)

```
case kStateFleeing:
  if (closest_agent == nullptr)
  {
    state_ = AgentState::kStateResting;
    target_entity_ = nullptr;
    start_rest_time_ = SDL_GetTicks();
  }
  else
  {
    if (closest_agent->size_ > size_)
    {
      target_entity_ = closest_agent;
    }
    else if (closest_agent->size_ < size_)
    {
      state_ = AgentState::kStateChasing;
      target_entity_ = closest_agent;
    }
  }
  break;
```

8)

```
case kStateResting:
  if (closest_agent != nullptr && closest_agent->size_ > size_)
  {
    state_ = AgentState::kStateFleeing;
    target_entity_ = closest_agent;
  }
  else
  {
    if ((SDL_GetTicks() - start_rest_time_) > kRestDuration * 1000.0f)
    {
      state_ = AgentState::kStateWorking;
    }
  }
  break;
```

9)

```
void Agent::update(u32 delta_time)
{
  switch (state_)
  {
    case kStateWorking:
      workingState(delta_time);
      break;

    case kStateChasing:
      MOV_tracking(delta_time);
      break;

    case kStateFleeing:
      MOV_tracking(delta_time, true);
      break;

    case kStateResting:
      MOV_stopped(delta_time);
      break;

    default:
      assert(false && "Agent update default case.");
      break;
  }
```

4) Los diferentes estados del agente listados en un enumerador.
5) Llamada a la función que actualiza los estados, comprobación de la distancia del agente más cercano y condiciones de cambio de estado para el estado Working.
6) Condiciones de cambio de estado para el estado Chasing.
7) Condiciones de cambio de estado para el estado Fleeing.
8) Condiciones de cambio de estado para el estado Resting.
9) Comportamiento de cada uno de los estados (workingState contiene un switch que, dependiendo del tipo de agente, llama a su movimiento).

Tecnologías emergentes de Inteligencia Artificial

Code-based Reasoning

Esta técnica permite a una máquina actuar como lo haría una persona humana basándose en experiencias pasadas. Ya ha sido utilizada, por ejemplo, en “Killer Instinct” para crear adversarios capaces de actuar tal y como tú mismo lo harías basándose en tus combates.
Esta técnica podría llegar a eliminar por completo la necesidad de jugar con y contra otros jugadores para divertirte en juegos online competitivos, ya que podrán programarse “bots” con comportamientos y razonamientos como los haría una persona real.

Hierarchical Task Network (HTN) Planning

En la industria de los videojuegos, pueden utilizarse técnicas de inteligencia artificial en otros campos aparte del propio videojuego. Técnicas de inteligencia artificial como el HTN Planning (utilizada en problemas de logística y coordinación militar) pueden ser muy útiles a la hora de planear y definir las tareas que conlleva la creación de cualquier videojuego, ya que son proyectos con muchas personas involucradas.
Si se utilizan técnicas como esta, podríamos ver un aumento de la calidad general de los videojuegos ya que se reduciría el tiempo y coste de producción.

Machine Learning

No es ningún secreto que una enorme cantidad de datos bien interpretados puede dar información muy valiosa, y eso es especialmente útil cuando diseñas un videojuego.
Utilizando algoritmos de machine learning con muchos datos (diferentes juegos, partidas, niveles, pruebas…) se puede determinar cómo es el flujo del nivel, dónde están las zonas conflictivas e incluso cuantificar la diversión. Teniendo eso en cuenta, cada vez se irán creando herramientas más potentes para los diseñadores de videojuegos, que creo que será el área del desarrollo de videojuegos que más avances tendrá en los próximos años gracias a la inteligencia artificial.

Bibliografía

Graft, K., 2015. *When Artificial Intelligence In Video Games Becomes...Artificially Intelligent*. [online] Gamasutra.com. Available at: <https://www.gamasutra.com/view/news/253974/When_artificial_intelligence_in_video_games_becomesartificially_intelligent.php> [Accessed 18 May 2020].

Thompson, T., 2017. *The Killer Groove: Shadow AI Of Killer Instinct*. [online] Medium. Available at: <https://towardsdatascience.com/killergroove-c4c606601a8a> [Accessed 18 May 2020].

Shaleynikov, A., 2018. *Integrating Machine Learning Into Game Development | Hacker Noon*. [online] Hackernoon.com. Available at: <https://hackernoon.com/integrating-machine-learning-into-game-development-c5a7f31ed839> [Accessed 18 May 2020].