Inteligencia Artificial para Videojuegos
Grado en Desarrollo de Videojuegos
Proyecto final
Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Proyecto: Gambito de Mus

Grupo 7: Tomás López Antón, David Godoy Ruiz, Eva Lucas Leiro, Jorge Bello Martín, Aitor García Casamayor

1. Introducción

"Ser un buen perdedor es aprender cómo ganar"

Hemos pensado que sería una buena idea programar una IA que juegue a un juego de cartas. Habíamos pensado en integrar algún tipo de **TCG** (Trading Card Game), pero, realmente no es muy original dentro del contexto de los videojuegos, dado a que hay muchos. Pero, ¿qué hay de los **juegos tradicionales**? No tienen mucha representación (salvo excepciones). Por lo tanto, hemos decidido integrar en nuestro proyecto una IA que sepa jugar al mus.

El mus es un juego de cartas complejo, cuyo principal atractivo son las **dinámicas y estrategias** que sus jugadores generan. Será interesante programar una IA capaz de identificar posibles jugadas y realizarlas con éxito.

2. Planteamiento del proyecto

Para realizar este proyecto, hemos decidido realizar varias **iteraciones** sobre la IA, planteándonos diferentes objetivos consecutivos.

La primera iteración consta de crear una simulación del mus, integrando la baraja española y las diferentes reglas para crear el flujo de juego en el que podrían jugar dos jugadores, uno contra

uno. Pero, obviamente, nuestra intención es que exista una IA contra la que jugar, no para jugar contra otro jugador. Por lo tanto, una vez integrado el juego de cartas, integraremos una IA que conozca y sepa aplicar diferentes jugadas y contra la que enfrentarse.

La segunda iteración, por lo tanto, se tratará de implementar el juego en parejas. El mus es un juego de cartas ampliamente conocido por jugar en parejas. La idea es que esta IA sepa interactuar e identificar las jugadas que hace el jugador (o IA) que tiene como compañero, actuando de tal manera que potencie y prediga las jugadas que haga el jugador.

Una vez llegado a este punto, lo que debería ocurrir es que pudiese existir una partida en la cual hay cuatro IA jugando con y contra las otras, ya que será capaz de realizar jugadas en ambas direcciones.

El proyecto será realizado con Unity, utilizando las herramientas que hemos aprendido en clase (Bolt, Behaviour Designer…) e integrando nuestros propios scripts y escenas.

3. Nuestro juego: Gambito de Mus

Como campo de pruebas para nuestra IA, realmente necesitábamos algo más que un pequeño entorno, porque al ser un juego de cartas necesitamos poder contemplar todos los casos posibles dentro del juego, siguiendo sus reglas. Por lo tanto, hemos escalado un prototipo a un juego en sí, que podemos jugar de principio a fin.

Como hemos comentado antes, para desarrollarlo hemos utilizado la herramienta de **Unity**, concretamente en su versión 2020.2.7f1. El juego consta con dos escenas principales: el menú inicial, en el cual elegimos si queremos jugar una partida contra dos IA (y con otra acompañándonos como equipo) o queremos ver a cuatro IA jugando contra ellas.

La siguiente escena es la más importante: la escena de juego. En ella se desarrolla toda la partida. Hemos utilizado el patrón *singleton* con un script `GameManager` para controlar el ciclo, funciones comunes y reglas del juego. Antes de comentar el ciclo de juego, hablaremos de la estructura de datos existente en el `GameManager` para llevar la lógica de juego.

En primer lugar, formando parte de nuestro `namespace IAV.G07.MUS` tenemos varios tipos enumerados y clases para contener datos. Estas son las siguientes:

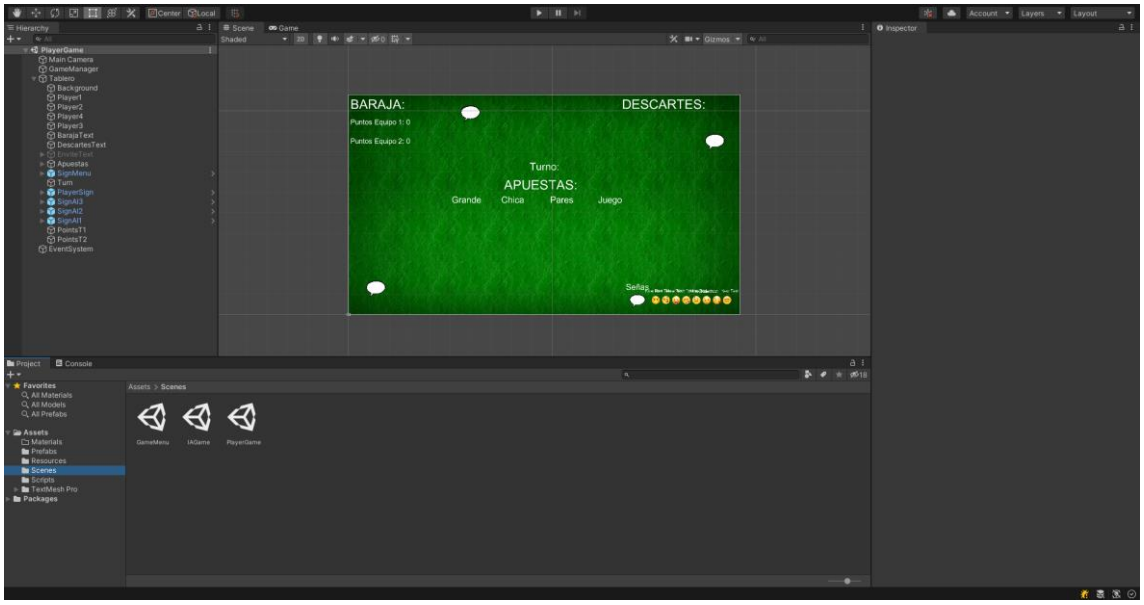
- `public enum Fase`: enum para controlar la fase en la que se encuentra la partida.
- `public enum CardType`: tipo enumerado de las cartas (Bastos, Espadas, Copas y Oros)
- `public enum SignEnum`: tipo enumerado para identificar qué seña ha realizado un jugador a su pareja.
- `public class Apuesta`: clase para guardar los datos de los envites: el equipo que lo realizó y el valor de la apuesta.
- `public class Card`: clase para guardar la información de las cartas: su `CardType` (palo), el valor numérico de la carta, el sprite que se renderizará en la partida, y una

variable booleana que sirve para identificar si el jugador quiere mandarla a la pila de descartes o no.

- `public enum Action`: tipo enumerado para identificar la acción que realiza un controlador.

Como hemos comentado, seguimos un patrón singleton. Esto quiere decir que tenemos una variable pública estática de tipo `GameManager` llamada `Instance` con la cual accederemos a la única instancia en partida del `GameManager`, para poder acceder desde otros componentes a sus métodos y variables públicas que necesitemos. Para controlar las cartas de la partida, tenemos dos pilas (tipo `Stack<Card>`) llamadas `_baraja` y `_descartes` (como sus nombres indican, para controlar qué y cuantas cartas hay en la baraja y en la pila de descartes). Además, guardamos las apuestas en una `List<Apuesta> _envites`. Posteriormente, disponemos de unas cuantas variables privadas para llevar cuentas y tener constancias de las últimas acciones realizadas por los jugadores, además de los puntos de cada equipo, y por último unas variables públicas para conectar con los elementos de la escena de Unity y mostrar la UI de la partida.

Hablando de la escena de Unity, está formada únicamente por elementos de tipo `Canvas`. Estos elementos que nos ofrece la API de Unity ofrecen mucha comodidad para mostrar texto por pantalla e imágenes estáticas, además de para controlar input de ratón. Dado que nuestro juego se basa casi exclusivamente de este tipo de interacciones y objetos en la escena, nos pareció lo más limpio y eficiente.



El ciclo de juego ha sido creado siguiendo un Enum llamado `Fase`, el cual contiene las diferentes fases de la partida, y mediante mensajes entre los controladores de los diferentes jugadores (principalmente gracias a otro Enum llamado `Action`) y el script `GameManager` identificamos qué se debe hacer en cada momento. Las fases de la partida son las siguientes:

- **Fase de mus (Fase.Mus)**: En esta fase, los jugadores deben decidir si quieren hacer mus o no. Hacer mus significa que quieres descartarte de algunas de tus cartas, y decir que no quieres hacer mus significa que quieres empezar las fases de apuestas. Para pasar a la fase de descartar, todos los jugadores deben decidir hacer mus consecutivamente, y desde que alguno de los cuatro jugadores decida no hacer mus, se comenzará la primera fase de apuestas. Se puede pasar de la fase de mus a la fase de descarte de manera infinita hasta que alguien decida no hacer mus.
- **Fase de descartes (Fase.Descartar)**: Si los cuatro jugadores deciden hacer mus, tendrán la opción de descartar 0,1,2,3 o 4 cartas. En esta fase se pregunta uno a uno qué cartas quiere o no quiere descartar, y en caso de hacerlo, las cartas seleccionadas irán a la pila de descartes. Posteriormente, el jugador robará de la baraja el mismo número de cartas que descartó. Si a la hora de robar cartas no quedan cartas en la baraja, se volverá a barajar la pila de descartes a la baraja, teniendo de nuevo la baraja llena (sin contar como es lógico las cartas que están en las manos de los jugadores). Una vez todos los jugadores deciden qué quieren hacer, se volverá a la fase de mus.
- **Fases de apuestas:"grande" (Fase.Grande), "chica" (Fase.Chica), "pares" (Fase.Pares) y "juego" (Fase.Juego)**: Estas son las fases de apuestas. En ellas, los jugadores deberán decidir si quieren realizar un envite de puntos o pasar de turno (un envite es una apuesta) basándose en los criterios de la fase. Una vez uno de los dos equipos realiza una apuesta, el siguiente equipo puede decidir subir la apuesta, pasar de ella o "ver" la apuesta. Todas estas acciones serán explicadas más adelante. Existen varios casos concretos:
    - Si todos los jugadores deciden pasar en lugar de envidar, se añade una apuesta base en la mesa con valor de 1 punto.
    - Si un jugador realiza un envite, y los dos jugadores del equipo rival deciden pasar a esta primera apuesta, se sumará un punto automáticamente al equipo que realizó el envite y se pondrá en la mesa una apuesta de valor 0.
    - Si un equipo A realiza una apuesta, el equipo contrario B la sube, y el equipo A decide pasar de dicha subida, este equipo B se quedará automáticamente con los puntos que había sobre la mesa antes de subir la apuesta y no se tendrán en cuenta durante el recuento final, modificando la apuesta de la mesa por una apuesta con valor de 0.

  En cualquier otro caso, las subidas o envites iniciales se quedarán en la mesa para el recuento final. Este funcionamiento es el mismo para todas las fases de apuesta, sólo diferenciándose en el criterio de evaluación dependiendo de en cuál nos encontremos, y se realizan en el siguiente orden: primero "grande", en segundo lugar "chica", en tercer lugar "pares" y por último "juego". Una vez pasado de la fase de "juego", se pasa a la fase de recuento de puntos. Los criterios de evaluación de puntos según la fase se comentarán en el siguiente punto.

- **Fase de recuento de puntos (Fase.Puntos)**: Una vez llegados a esta fase, en la mesa habrá cuatro envites o apuestas. En la fase de puntos se tendrán en cuenta todos (a excepción de las apuestas con un valor de 0, las cuales son añadidas en el caso concreto comentado anteriormente) los envites guardados en la lista _envites. Se realizarán los recuentos de puntos siguiendo el orden de las fases de apuestas comentados anteriormente. En este momento, todos los jugadores muestran sus manos y comienza la evaluación de las mismas. Primero se evaluará las manos de cada equipo, y

posteriormente se compararán las mejores manos de cada equipo entre ellas. Los criterios de evaluación son los siguientes:

- Para el envite de la fase “grande”, gana la mano que tenga la carta más alta. En caso de empate, gana el equipo 1.
- Para el envite de la fase “chica”, gana quien tenga la carta más baja. En caso de empate, ocurre igual que en la fase “grande”.
- Para el envite de la fase de “pares”, gana la mejor mano, entendiendo que unos “Duples” son mejores que un “Trío (“medias”)”, y que éste es mejor que una “Pareja”.
- Para el envite de la fase de “juego”, ganará la mano que tenga unas cartas con un valor numérico más cercano a 31.

Una vez comparadas las manos, se sumarán los puntos correspondientes a cada envite a cada uno de los equipos ganadores de cada ronda. Ganará la partida quien tenga más puntos de los dos equipos.

- **Fase final (Fase.Final):** Una vez el bucle ha llegado a la fase final, simplemente mostramos un mensaje de victoria y volvemos a llevar al jugador al menú principal.

Entre los métodos implementados en GameManager, cabe destacar los siguientes:

- `Update`(): en el método `Update`() comprobamos en cada frame en qué situación se encuentra la partida: comprobamos la fase actual entre las anteriormente mencionadas, y además comprobamos si el jugador del turno actual ha acabado o no de decidir sus acciones.
- `Game`(): en este método se evalúan las acciones que un controlador realiza en las fases de apuesta. Por ello, controla todas las posibles situaciones de la manera más generalizada posible para los casos especiales y los comunes que pueden ocurrir.
- `CompareHands(List<Card>first, List<Card> second, Envite e)`: en este método realizamos todos los cálculos matemáticos y comparaciones necesarias para evaluar dos manos (`first y second`) según los criterios de evaluación del `Envite e` (que se trata de un indicador de la fase en la que nos encontramos).
- `Barajar(Stack<Card>b)`: utilizando el algoritmo de Fisher-Yates shuffle, barajamos de manera eficiente una pila de cartas (la baraja o la pila de descartes).

Los controladores se controlan mediante herencia. Tenemos una clase base `Player`, de la cual heredan los dos controladores principales: `UserPlayer`, que se trata del controlador humano, y `JoaquinPlayer`, que se trata de nuestra IA. En la clase base `Player` contenemos la información y métodos comunes para ambos: la mano del jugador, las señas que han conseguido identificar en el resto de jugadores, la fase en la que se encuentra, la apuesta que deciden tirar a la mesa, y métodos para dibujar y ordenar sus manos.

En cuanto a las acciones que pueden realizar los controladores, éstas se comparten independientemente de qué tipo de controlador sean. La única diferencia de los controladores es el cómo realizan dichas acciones (mediante un input si es un controlador humano, o mediante evaluaciones matemáticas si se trata de la IA). Las acciones son las siguientes, guardadas en un tipo enumerado llamado `Action`:

- Acción de espera (`Action.Inicial`): acción de espera inicial, en la cual no ha realizado ninguna acción aún.
- Acción de envidar `(Action.Envidar)`: acción para realizar una apuesta.
- Acción de pasar (`Action.Pasar`): acción para pasar de una apuesta.
- Acción de ver: (`Action.Ver`): acción para ver una apuesta.
- Acción de subir (`Action.Subir`): acción para subir una apuesta.

El controlador realizará una de las acciones mencionadas y se las comunicará al `GameManager`, manteniendo al tanto también de si ha acabado de decidirse en los momentos de espera o no. Será este el que, según la acción y teniendo control de en qué turno de la partida se encuentra, realizará diferentes secuencias según las acciones elegidas por los controladores.

El controlador humano podrá escoger sus acciones según la siguiente lista de inputs:

- Tecla N: decidir no hacer mus en la fase de mus.

- Tecla S: subir una apuesta o decidir hacer mus en la fase de mus.
- Tecla E: realizar un envite.
- Tecla P: pasar de un envite.
- Tecla V: ver una apuesta.
- Ratón: seleccionar una seña en pantalla o hacer click en la caja de texto que aparece en pantalla cuando decide realizar una apuesta.
- Teclado numérico: para escribir una apuesta (cualquier combinación de teclas)
- Teclas 1/2/3/4: teclas para seleccionar las cartas que quiere descartarse en la fase de descarte.

Por último, cabe destacar el sistema de señas. Se trata de una recreación de las muecas y señas que se realizan durante una partida de mus, para decirle a tu compañero sin hablar qué buenas o malas combinaciones de cartas tienes en tu mano, y con ello tomar decisiones de apuestas (incluso faroles). Las señas son los siguientes:

- "Dos reyes": morderse el labio inferior.
- "Tres reyes": morderse el labio inferior de manera lateral.
- "Dos ases": sacar la lengua por el centro.
- "Tres ases": sacar la lengua por un lateral de la boca.
- "Duples"(dos pares): levantar las cejas.
- "Treinta y una"(31 puntos): guiñar el ojo.
- "Solomillo" (tres reyes y un as): dar un beso al aire.
- "Ciego" (no llevar nada): cerrar los ojos.

4. Joaquín, nuestro jugador controlado por IA

Joaquín es nuestro jugador de mus controlado por IA. Al ser el mus un juego de información incompleta al no saber ni las manos de los oponentes ni la del compañero, Joaquín debe hacer estimaciones de las manos posibles de sus rivales y jugar en consecuencia. Además, cuando Joaquín recibe su mano, lanza una seña a su compañero acorde con la mano que le ha tocado

Joaquín tiene 3 intervalos para cada juego: un intervalo de "mala mano", con el que pedirá mus, no envidará cuando le toque ni aceptará un envite de los rivales; un intervalo de "buena mano", con la que cortará mus, envidará en su turno y verá envites no muy arriesgados*. Por último tiene un intervalo de "muy buena mano", con el que hará envites más cuantiosos, subirá envites normales y aceptará envites más arriesgados. Aun así, ya que el mus es un juego de engaño además de un juego de azar, Joaquín tiene una probabilidad de que, aunque tenga una mala mano, envide en una ronda, lo que le hace más impredecible y más parecido a un jugador humano.

A la hora de echarse mus, Joaquín tirará todas las cartas de su mano que no sean reyes, incluyendo parejas bajas ya que tras un mus no son tan valiosas.

Para analizar las manos, Joaquín tiene una lista de todas las cartas que son posibles en las manos de sus rivales. Esto no incluye las cartas de su mano, ni las cartas que tanto él como sus rivales hayan descartado previamente durante las fases de mus. Esto provoca que si en una ronda se dan varios muses Joaquín considera que es más probable que sus rivales tengan una mano mejor, lo que hace que no se arriesgue con manos con las que en una primera ronda podría haber tirado un envite.

*Joaquín hace esto con todos los juegos menos con chica, ya que de forma normal en el mus un jugador con una mano de chica prácticamente da por perdidos los juegos de grande, pares y juego.

5. Conclusiones finales

Realizar este proyecto ha sido una experiencia bastante enriquecedora. Realizar un juego desde cero siguiendo un juego de cartas ha sido una buena experiencia ya que al tener un diseño y unas reglas cerradas, el proceso de trabajo se basó en buscar la mejor manera de representar tanto la estructura de datos del juego como mostrarlo de manera jugable, cómoda y entendible mediante el motor Unity. Este motor nos ha facilitado muchas cosas y nos ha dado herramientas muy útiles para realizar el prototipo y el juego en sí mismo. Además, realizar un proyecto acerca de un juego que siempre ha estado en la cultura de nuestros entornos resulta más fácil y gratificante. Cabe destacar que la inteligencia artificial que hemos desarrollado sigue unas jugadas típicas del juego de cartas y eso ha ayudado mucho a su diseño y concepción inicial, debido a que las acciones y evaluaciones que queríamos que realizara fueron basadas en las jugadas y lógicas humanas detrás del propio juego del mus.

Aunque no hayamos podido implementar ciertas cosas que nos hubiera gustado (comportamiento dinámico, máquinas de estado, interpretación de señas…), estamos muy satisfechos con el resultado final hasta el punto que consideramos una IA que podría existir en un juego sencillo o como minijuego dentro de un juego mayor, y ser entretenido para el jugador.

6. Referencias y ampliaciones

- Información en Wikipedia sobre el mus
- Bolt, Visual Scripting
  https://unity.com/es/products/unity-visual-scripting
- Fisher-Yates shuffle
  http://en.wikipedia.org/wiki/Fisher-Yates_shuffle
- Opsive, Behavior Designer
  https://opsive.com/assets/behavior-designer/
- Unity 2018 Artificial Intelligence Cookbook, Second Edition (Repositorio)
  https://github.com/PacktPublishing/Unity-2018-Artificial-Intelligence-Cookbook-Second-Edition
- Unity Artificial Intelligence Programming, Fourth Edition (Repositorio)
  https://github.com/PacktPublishing/Unity-Artificial-Intelligence-Programming-Fourth-Edition