Procesamiento de imágenes en juegos de Atari para facilitar el aprendizaje reforzado

Mateos Manjón, Daniel

Curs 2020-2021

Director: ANDERS JONSSON

FEDERICO SUKNO

GRAU EN ENGINYERIA DE SISTEMES AUDIOVISUALS

upf. Universitat Pompeu Fabra Barcelona
Escola d'Enginyeria

Treball de Fi de Grau

Procesamiento de imágenes en el juego Freeway de Atari para facilitar el aprendizaje reforzado

TRABAJO DE FINAL DE GRADO

Daniel Mateos Manjón

Directores: Anders Jonsson
Federico Sukno

Grado en Ingeniería en Sistemas Audiovisuales

Curso 2020-2021

Universitat Pompeu Fabra Barcelona
Escola d'Enginyeria

A mi familia.

AGRADECIMIENTOS

Agradezco este trabajo sobre todo a mis tutores Anders Jonsson y Federico Sukno por toda su ayuda y apoyo en la creación del mismo. También mi familia, que ha estado a mi lado en todo momento y me ha aportado tranquilidad. Agradezco a mis amigos por ayudarme a desconectar en momentos de agobio y a motivarme a seguir trabajando cuando lo necesitaba. Y finalmente, agradezco a mis abuelos por su enorme apoyo y afecto que me generaba una actitud positiva.

Resumen

En este proyecto utilizaremos técnicas de procesamiento de imágenes para obtener información simbólica de frames en juegos de la Atari 2600, en concreto, centrado en el juego Freeway. Para ello, utilizaremos una estructura llamada Arcade Learning Environment (ALE), que permite emular juegos de la Atari 2600 en ordenador y obtener información como la imagen del frame y la recompensa para cada estado. Esta información más relevante será obtenida mediante la aplicación MATLAB® con el ya mencionado procesamiento de imágenes. Finalmente, un agente que recibirá la información decidirá qué acción tomar para el siguiente estado de la ALE mediante un sistema de aprendizaje reforzado. De este modo reducimos de manera drástica la cantidad de información que recibe para decidir qué acción tomar para obtener la recompensa más elevada mientras juega.

Resum

En aquest projecte farem servir tècniques de processament d'imatges per obtenir informació simbòlica de frames en jocs de l'Atari 2600, en concret, centrat en el joc Freeway. Per a això, utilitzarem una estructura anomenada Arcade Learning Environment (ALE), que permet emular jocs de l'Atari 2600 en ordinador i obtenir informació com la imatge del frame i la recompensa per a cada estat. Aquesta informació més rellevant serà obtinguda mitjançant l'aplicació de MATLAB® amb el ja esmentat processament d'imatges. Finalment, un agent que rebrà la informació decidirà quina acció prendre per al següent estat de l'ALE mitjançant un sistema d'aprenentatge reforçat. D'aquesta manera reduïm de manera dràstica la quantitat d'informació que rep per decidir quina acció prendre per obtenir la recompensa més elevada mentre juga.

Abstract

In this project we will use image processing techniques to obtain symbolic information about frames in Atari 2600 games, specifically, focused on the Freeway game. To do this, we will use a structure called Arcade Learning Environment (ALE), which allows us to emulate Atari 2600 games on a computer and obtain information such as the frame image and the reward for each state. This most relevant information will be obtained through MATLAB®, with the aforementioned image processing. Finally, an agent who will receive all the information will decide what action to take for the next ALE state using a reinforced learning system. In this way, we drastically reduce the amount you receive for the network when deciding what action to take to obtain the highest reward while playing.

ÍNDICE

LISTA DE FIGURAS

1. INTRODUCCIÓN

En este primer apartado se introducirá la idea de este trabajo, mostrando sus motivaciones y objetivos, así como una visión general del mismo para comprender la relación entre posteriores apartados.

1.1 Motivación

Los avances tecnológicos en el campo de la inteligencia artificial a lo largo de los años han permitido la creación de agentes capaces de superar el nivel humano en muchos juegos. En 1997, una computadora de IBM a la que denominaron Deep Blue, ganó por primera vez al entonces campeón mundial de ajedrez, Gary Kasparov, tras un encuentro a 6 partidas. Este evento no fue más que el principio, y actualmente encontramos IAs que son capaces de jugar a juegos mucho más complejos, dando paso a la inteligencia artificial en el mundo de los videojuegos.

Una de las videoconsolas más famosas y más vendidas de los 80 fue la Atari 2600, dado su popularidad y sus más de 500 juegos esta se convirtió en un icono. Esta videoconsola y sus juegos han servido de base para el estudio de inteligencias artificiales en el campo de la investigación y el estudio.

En la Universidad de Alberta un grupo de ingenieros ha desarrollado una estructura llamada Arcade Learning Environment con la finalidad de facilitar el estudio de inteligencias artificiales. Está estructura es conocido popularmente como ALE y será explicada en este mismo proyecto. La ALE utiliza el emulador de Stella para jugar a juegos de la Atari 2600 a la vez que proporciona información de la memoria RAM y acceso a la imagen de los frames en tiempo real, lo cual permite un aprendizaje constante del agente.

A diferencia de los agentes qué son entrenados en entornos diseñados específicamente para ellos y su aprendizaje, la ALE permite un desarrollo en un entorno que originalmente fue creado con la finalidad de entretener a las personas y por ello nos permite evaluar agentes en entornos más genéricos. Aparte de la investigación con respecto al aprendizaje reforzado, la motivación principal de este proyecto es la investigación del uso del procesamiento de imágenes a la hora de aportar información al aprendizaje reforzado.

1.2 Objetivos

La idea es aplicar procesamiento de imágenes a los frames de la Atari 2600, reduciendo la cantidad de información de estos. La información que llega al agente de este modo es mucho más reducida que toda aquella que aporta el frame en sí de todos los píxeles, pero que a su vez esta poca información será mucho más importante.

El primer objetivo es encontrar una implementación de técnicas de procesamiento de imágenes que sea capaz de obtener la información más relevante de un frame.

El segundo, conseguir una relación clara entre el campo de procesamiento de imágenes digitales y el campo de la inteligencia artificial.

El objetivo principal es desarrollar un agente capaz de entrenar mediante el aprendizaje reforzado con dicha información reducida como entrada.

Por último, se comprobará si el aprendizaje reforzado da buenos resultados teniendo menos información como input observando el sistema de recompensas, demostrando así su potencial a la hora de deshacerse de información irrelevante que puede ralentizar el aprendizaje.

1.3 Visión general del informe

En este informe se incluye una descripción detallada de los diversos procesos necesarios para obtener los objetivos mencionados en el apartado anterior.

En la sección posterior mencionaremos algunos proyectos relacionados con este trabajo por lo que respecta a la inteligencia artificial en videojuegos.

En la sección 3 nos centraremos en la consola en la que este proyecto está basado, la Atari 2600. Introduciéndonos en la historia de las videoconsolas y explicando su posición en ese mundo. También explicaremos algunas de sus características por las que se convirtió en una videoconsola icónica de los 80. Presentaremos "the Arcade Learning Environment"(ALE), una estructura capaz de emular el funcionamiento de la Atari 2600 con la capacidad de obtener información de ciertos videojuegos, lo cual será muy necesario para este proyecto por su relación con las dos secciones posteriores. Explicaremos como instalar y compilar la ALE, sus características más importantes y como obtener la información necesaria para el procesamiento de imágenes. Finalmente hablaremos de sus videojuegos, de los que destacaremos *Freeway*, el cual ha sido elegido para este proyecto.

Una de las características de la ALE mencionada en párrafo anterior, es su capacidad para obtener información de los juegos emulados, como por ejemplo la imagen de cada frame en cada estado. En la sección 4 tratamos el procesamiento de imágenes enfocado en obtener la información más relevante, tal y como se explica en los objetivos anteriores. En esta sección se explicará que es el procesamiento de imágenes enfocado en imágenes digitales, y mencionaremos algunas de las aplicaciones por las que se usa el procesamiento de imágenes. Nos centraremos en explicar principalmente los métodos utilizados en este proyecto para extraer la información más relevante de los frames, tales como el *optical flow (OF)*. También explicaremos como relacionar los resultados obtenidos por el OF con la ALE a la hora de organizar la información deseada del juego. Como último subapartado de la sección, mencionaremos MATLAB, como el programa elegido para implementar este procesamiento.

En la sección 5 se explica el funcionamiento del aprendizaje reforzado a la hora de entrenar a un agente, con la información obtenida del frame gracias al procesamiento de imágenes la cantidad de variables que este recibe es drásticamente menor, pero simultáneamente más relevante.

Todos estos procesos son implementados y explicados desde el punto de vista práctico en la sección 6 donde mencionaremos algunos problemas y las soluciones que hemos tomado mientras el proyecto estaba en proceso.

Comentaremos y evaluaremos los resultados de la implementación en la sección 7.

Finalmente, para concluir el informe se incluye una conclusión en la octava sección donde se mencionarán los aspectos mejorables del proyecto de cara a obtener unos mejores resultados.

2. TRABAJOS RELACIONADOS

La compañía inglesa *DeepMind*[1] trabaja con inteligencias artificiales, esta fue creada en 2010 y actualmente es propiedad de *Google*[2], ya que fue comprada en 2014 por *Alphabet Inc.*[3]. La compañía ha creado diversas redes neuronales capaces de aprender a jugar de una manera similar a la de las personas y entre ellas destacamos Agent57.

Agent57 es un agente que utiliza el aprendizaje reforzado para jugar a juegos de una videoconsola conocida como Atari 2600, de la cual hablaremos más adelante en este trabajo. La compañía ya había diseñado anteriormente agentes capaces de jugar a varios videojuegos de la consola, pero Agent57 el primer agente que utiliza aprendizaje reforzado profundo para aprender a jugar a 57 videojuegos distintos.

El proyecto de DeepMind, está en gran parte relacionado este trabajo. DeepMind utiliza la misma estructura como base para jugar al Atari que la que se utiliza en este proyecto, la cual es explicada en un apartado posterior. Principalmente, a gran diferencia es que la idea de este trabajo es simplificar el aprendizaje reforzado mediante las técnicas de procesamiento de imágenes, mientras que Agent57 busca ser un agente capaz de superar a las personas en una gran variedad de videojuegos.

[1] https://deepmind.com/
[2] https://www.google.com/
[3] https://abc.xyz/

3. VIDEOCONSOLAS Y LA ATARI 2600

Una videoconsola es un sistema electrónico que ejecuta videojuegos, cuya principal finalidad es entretener y divertir mediante la ejecución de videojuegos. Los videojuegos son, como indica parcialmente el nombre, juegos electrónicos en los que jugadores manipulan mediante controladores un dispositivo de video llamado plataforma. Estos pueden ser accedidos por el sistema mediante discos, tarjetas de memoria, cartuchos o cualquier otro tipo de almacenamiento digital.

Similares a los ordenadores, las videoconsolas pueden encontrarse de diferentes formas y tamaños dependiendo de la marca y de las características para las que fueron creadas (Figura 1), algunas de las compañías más famosas son *Nintendo, PlayStation y Microsoft*.

Figura 1: Ejemplos de videoconsolas

3.1 Historia de las videoconsolas

Dada la gran cantidad de modelos de videoconsolas y la diferencia de calidad debido a las mejoras que hay entre las mismas, estas fueron divididas en grupos según su año de salida, dichos periodos fueron denominados generaciones [2]. La primera videoconsola que salió en venta de cara al público fue en 1972, hasta entonces ya existía un concepto similar a lo que hoy en día llamamos videojuegos, pero estaba destinado únicamente a fines de estudio de desarrollo e investigación. Entonces, es a partir de ese año que las videoconsolas llegaron a los hogares de las personas, dando inicio a la primera generación. En los siguientes subapartados destacaremos características y ejemplos de cada generación.

Primera generación (1972-1976)
La primera generación empezó en 1972 con la salida de *Magnavox Odyssey* en Estados Unidos, proporcionando la oportunidad de jugar desde casa a juegos que hasta entonces solo se podía en salas recreativas. Se vendieron cerca de unas 100.000 unidades a unos 100 dólares cada una. Funcionaba con juegos insertados como cartuchos y había un total de 28 juegos distintos, pero bastante simples ya que la consola no disponía de procesador o memoria. Más tarde salió a la venta la *Atari pong* que resultó ser un éxito e influenció a muchas otras empresas a crear sus propias videoconsolas, pero estas

nunca llegaron a superarla. Estas videoconsolas tenían unos gráficos muy simples y generalmente eran en blanco y negro, además de no soler disponer de audio.
En estas generaciones la extracción de información era muy simple, ya que los juegos solo contenían 2 colores por pantalla y se basaban en entornos de 2D.

Figura 2: Magnavox Odyssey

Figura 3: Atari pong

Segunda generación (1976-1983)
Tras una caída desde el pico de la generación anterior, la compañía Atari[1] saco al mercado su nueva videoconsola llamada *Atari VCS*, más popularmente conocida como *Atari 2600*. La gran diferencia entre esta consola y las anteriores es que ahora disponían de memorias ROM en los cartuchos, permitiendo la existencia de juegos de mejor calidad y más complejos. También se incluyeron los joysticks, permitiendo una interacción mejor entre el jugador y el dispositivo. En esta generación se destaca también la aparición de la primera videoconsola creada por Sega[2], llamada *Sega SG 1000* que se vendió exclusivamente en Japón en 1981. Para estas consolas la extracción de información de los juegos también es sencilla al usar pocos colores por pantalla en juegos de 2D.

Figura 4: Atari VCS

Figura 5: Sega SG 1000

Tercera generación (1983-1987)
Del mismo modo que la generación anterior, la tercera generación experimento una gran caída qué freno a las compañías americanas, dando así una oportunidad a las compañías japonesas. La popularidad de la *Sega SG 1000* y el lanzamiento de la *Nintendo Entertainment System (NES)* en Estados Unidos ayudo a que la industria recuperará fuerza. En esta generación, la llegada de los 8 bits aumentó la resolución y la cantidad de colores en los juegos. También se introdujeron los controladores direccionales.

Figura 6: Nintendo Entertainment System

Cuarta generación (1987-1993)
Las compañías de Nintendo y Sega compiten entre sí por dominar el mercado creando nuevas consolas como la *Super Nintendo Entertainment System (SNES)* y la *Sega Mega Drive*. Pasamos de 8 a 16 bits, lo cual conlleva una nueva mejora en la calidad de los juegos al permitir almacenar más datos. En esta generación se permite utilizar periféricos para guardar videojuegos en CDs por primera vez.

Figura 7: Super Nintendo Entertainment System

Figura 8: Sega Mega Drive

Quinta generación (1993-1998)

En esta generación la capacidad de almacenaje de las ROM aumenta hasta los 650 MB gracias a los CDs. Con un tamaño tan grande empezamos a ver juegos con modelos 3D con gráficos poligonales complejos, que dejaban atrás el limitado mundo 2D. la compañía Sony lanzó al mercado su primera videoconsola llamada *PlayStation* y resultó ser un éxito vendiendo más de 100 millones de unidades, siendo la más popular con diferencia. En 1996 salió en Japón la *Nintendo 64*, llamada así por su procesador de 64 bits. La consola disponía de menos almacenaje de memoria, pero se compensaba con sus rápidos tiempos de carga. A partir de esta generación la extracción de información de los juegos se empieza a complicar, ya que estos comienzan a utilizar entornos 3D, de los cueles es más complejo extraer información del entorno.

Figura 9: PlayStation

Figura 10: Nintendo 64

Sexta generación (1998-2005)

En la sexta generación estandariza el CD como medio de distribución de de juegos. Nintendo saca la *GameCube*, Microsoft la *XBOX* y Sega la *Sega's Dreamcast*, pero de nuevo Sony vuelve a ser el icono popular al poner a la venta la *PlayStation 2* y la *PlayStation Portable (PSP)*. En esta generación vemos los primeros juegos online y destacamos una mejora de 64 a 128 bits, aunque no presento una gran diferencia con respecto al cambio de la generación anterior.

Figura 11:PlayStation 2

Séptima generación (2005-2012)
La resolución de la pantalla subió a HD en esta generación, los gráficos para consolas como la *XBOX 360* y la *PlayStation 3* fueron mejorados a nivel de CGI cinematográfico. En el 2006 Nintendo estrenó su nueva videoconsola, la *Wii*, siendo la primera consola en utilizar el concepto de mando inalámbrico para detectar los movimientos de los jugadores en juegos, los cuales principalmente eran de modo multijugador. Por lo general todas estas videoconsolas ya tienen acceso a Internet y la capacidad de reproducir contenido multimedia HD.

Figura 12: Wii

Octava generación (2012-2020)
La octava generación se caracteriza el uso de Internet como eje central, creando así un dispositivo multimedia con diversas funciones aparte de sistema de juego. Está generación comenzó oficialmente con el lanzamiento de la *Wii U* en 2012 y más tarde aparecerían otras como la *PlayStation 4* y la *XBOX ONE*. En esta generación también aparece un gran número de videoconsolas portátiles y a su vez otros dispositivos como los teléfonos inteligentes y los ordenadores son mejorados para soportar videojuegos al nivel de presentar una amenaza competitivamente hablando.

Novena generación (2020-actualidad)
A lo largo de 2019 tanto Microsoft como Sony anunciaron nuevas videoconsolas para 2020, la *XBOX X|Y* y la *PlayStation 5*. Estos sistemas son capaces de reproducir contenidos de una resolución de hasta 8K, siendo revolucionarios con semejante potencia, tal y como Andrea Rodríguez menciona en su artículo de EL COMERCIO [3]

"La irrupción del streaming, los servicios de contenido bajo demanda y la potencia técnica cercana a computadores de alta gama son aspectos que han llegado para quedarse."

Figura 13: PlayStation 5

3.2 La Atari 2600

La Atari 2600 o también conocida como Atari VCS (Video Computer System) es una videoconsola icónica de los años 80 creada por la compañía Atari que se incorporó al mercado en 1977, pero no llego a España hasta 1978. La compañía se encontraba en declive desde su anterior éxito, el *Pong*[4], y esta videoconsola presentó su nuevo salto a la fama en el mundo de los videojuegos.

La consola se vendía acompañada de un par de *joystiks* y otro par de *paddles* como controles para la misma. Algunas de las versiones, sobre todo las primeras, se vendían incluyendo el juego *Combat*.

Figura 14: Fotografía de la Atari 2600

La Atari 2600 dispuso de más de 500 juegos distintos, aunque muchos de ellos eran muy similares ya que compartían gran parte de las mecánicas. Por otra parte, algunos de estos juegos resultaron ser un éxito y de los más importantes de su generación, que

[4] Pong: juego exitoso de la Atari original

influenciaron al resto hasta el punto que a día de hoy siguen siendo considerados obras maestras. Ejemplos de estos juegos son *Pac-Man*, *Pitfall*, *Space Invaders* y *Donkey Kong* entre otros [4].

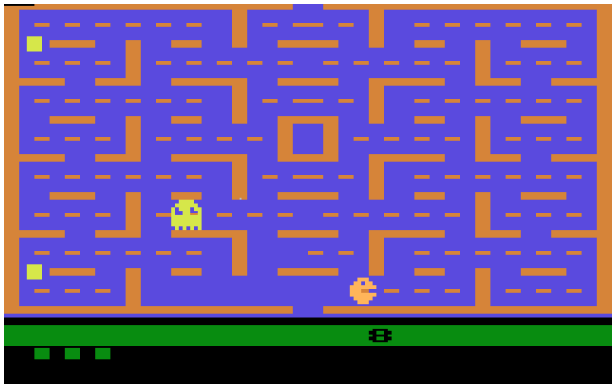

Figura 15: Juego Pac-Man de la Atari 2600

3.4 The Arcade Learning Environment

Para poder utilizar los juegos de la Atari 2600 como base para nuestro proyecto necesitamos una estructura no solo con la capacidad de emular la videoconsola, sino que también capaz de aportar información específica de cada estado del juego. En la Universidad de Alberta[5] en Canadá se desarrolló una plataforma llamada Arcade Learning Environment (ALE), donde desarrolladores pueden diseñar y evaluar inteligencias artificiales en multiples juegos[5].

La ALE utiliza el emulador de código abierto Stella[6] como base para ejecutar ROMs de videojuegos de la Atari 2600. Permite al usuario interactuar con la Atari 2600 recibiendo como input de entrada los movimientos del *joystik* y proporciona como salida una imagen del frame para cada estado en tiempo real o información de la RAM. Uno de los motivos por los que la ALE fue elegida para este proyecto, al igual que para otros muchos destinados al desarrollo de agentes de IA, es su capacidad para poder obtener información de las recompensas de cada estado de la RAM del juego, información que será necesaria para futuras partes de este proyecto relacionadas con el aprendizaje reforzado.

3.4.1 Características de la ALE

Cada imagen obtenida de la pantalla de la ALE tiene por defecto un tamaño de 210 pixeles de ancho por 160 de alto proporcionados en una matriz de 2D de pixeles de 7 bits.

Como hemos mencionado la ALE recibe como input un movimiento de *joystik*, el cual tiene 4 direcciones (arriba, abajo, derecha e izquierda) además de dos botones (agacharse y disparar), pero estas no son las únicas opciones, puesto que se pueden realizar diversos movimientos simultáneamente en uno mismo, por ejemplo, mover el *joystik* en diagonal significaría moverse en ambos ejes como sería en el caso de *arriba-*

[5] https://www.ualberta.ca/index.html
[6] https://stella-emu.github.io/

izquierda, o también se puede pulsar el botón de disparar mientras realizas un movimiento tal que *arriba-izquierda-disparar*. A cada una de las combinaciones se les asigna un valor y este es el que la ALE interpreta en función de la siguiente tabla traducida del *Arcade Learning Environment Technical Manual (v.0.6.0)* [6]:

sin acción (0)	disparar (1)	arriba (2)
derecha (3)	izquierda (4)	abajo (5)
arriba-derecha (6)	arriba- izquierda (7)	abajo-derecha (8)
abajo- izquierda (9)	arriba-disparar (10)	derecha-disparar (11)
izquierda-disparar (12)	abajo-disparar (13)	arriba-derecha-disparar (14)
arriba-izquierda-disparar (15)	abajo-derecha-disparar (16)	abajo-izquierda-disparar (17)

Adicionalmente para controlar a un segundo jugador en los juegos multijugador sus acciones se seleccionan como el valor de la tabla anterior sumándole 18.
Otras acciones extras para regular la ALE son:

resetear (40)	guardar-estado (43)
cargar-estado (44)	resetear el sistema (45)

Donde la acción resetear con valor 40 simula el botón de encendido y apagado de la Atari 2600.

3.4.2 Instalación de la ALE

ALE es un software de código abierto gratuito escrito en C++, pero dispone de diversas interfaces que permiten manipularlo desde otros lenguajes de programación por motivos de preferencia del usuario. Todas las versiones están disponibles en el repositorio de GitHub[7] de /mgbellemare en:

https://github.com/mgbellemare/Arcade-Learning-Environment

Se recomienda la instalación sobre un sistema operativo Linux[8] como puede ser *Ubuntu* y se requiere de las siguientes herramientas:

- **Compilador de C++**

- **Cmake**
  Una herramienta multiplataforma de generación o automatización de código. Podemos obtenerla con el siguiente comando en la terminal:

```
> sudo apt-get -y install cmake
```

- **Make**

Una herramienta capaz de generar ejecutables a partir de proyectos .cpp con sus Makefiles. Podemos instalar Make con el siguiente comando:

```
> sudo apt-get -y install make
```

[7] https://github.com/
[8] https://www.linux.org/

- **Soporte de audio y pantalla SDL**
  Las librerías de Simple DirectMedia Layer[9] no son obligatorias, pero sirven para proporcionar funciones básicas de dibujo en 2D y gestión de audio para programación en C, permitiendo observar los frames de la ALE en tiempo real. Instalamos la librería por terminal con:

```
> sudo apt-get install libsdl1.2-dev
```

Una vez dispones de las herramientas mencionadas puedes empezar a compilar e instalar la ALE, primero descargas el repositorio y desde una terminal accedes a la carpeta base donde has guardado la ALE (por defecto llamada *Arcade-Learning-Environment*) y segundo debes introducir los siguientes comandos en la terminal:

```
> mkdir build && cd build
> cmake ../ -DCMAKE_BUILD_TYPE=Release
> cmake --build . --target install
```

Utilizando Cmake podemos indicar que librerías deseamos. En nuestro caso indicamos que deseamos la de SDL para visualizar el juego, ya que suele estar desactivada por defecto, y genere los proyectos de ejemplo de los archivos `.cpp` incluidos.

```
> cmake -DUSE_SDL=ON -DBUILD_CPP_LIB=ON -DBUILD_EXAMPLES=ON
```

Tras este paso ya disponemos de ALE compilada en nuestro sistema. A la hora de seleccionar un juego hemos de indicar dos argumentos: el tipo de controlador (*fifo* por defecto) y si deseamos ver la pantalla del emulador de la Atari 2600. En una terminal en la carpeta base de la ALE ponemos el siguiente comando:

```
> ./ale –game_controller fifo –display_screen true [GAME]
```

Donde `[GAME]` es el archivo ROM del juego deseado y que sebe estar situado dentro de la carpeta de la ALE con su camino correspondiente.

Figura 16:Frame de la ROM de Skiing

[9] https://www.libsdl.org/

3.4.3 Extracción de información de la ALE

En la carpeta descargada de la ALE, si vamos a la subcarpeta de `doc/examples` encontraremos un archivo de C++ llamado `videorecordingexample.cpp` en el cual se encuentra un código que al ser ejecutado guarda en una carpeta todos los frames del juego en tiempo real por cada movimiento del jugador. Podemos ejecutar el programa con el siguiente comando en la terminal desde la carpeta base de la ALE:

```
> doc/examples/videorecordingexample [GAME]
```

Donde [GAME] es el archivo ROM del juego deseado.

En el código `.cpp` observamos que se genera una carpeta llamada `record` en la carpeta base de la ALE y cada vez que se introduce una acción/movimiento al jugador se guarda la imagen del estado en la carpeta con su número de frame como nombre, tal y como se observa en la figura inferior.

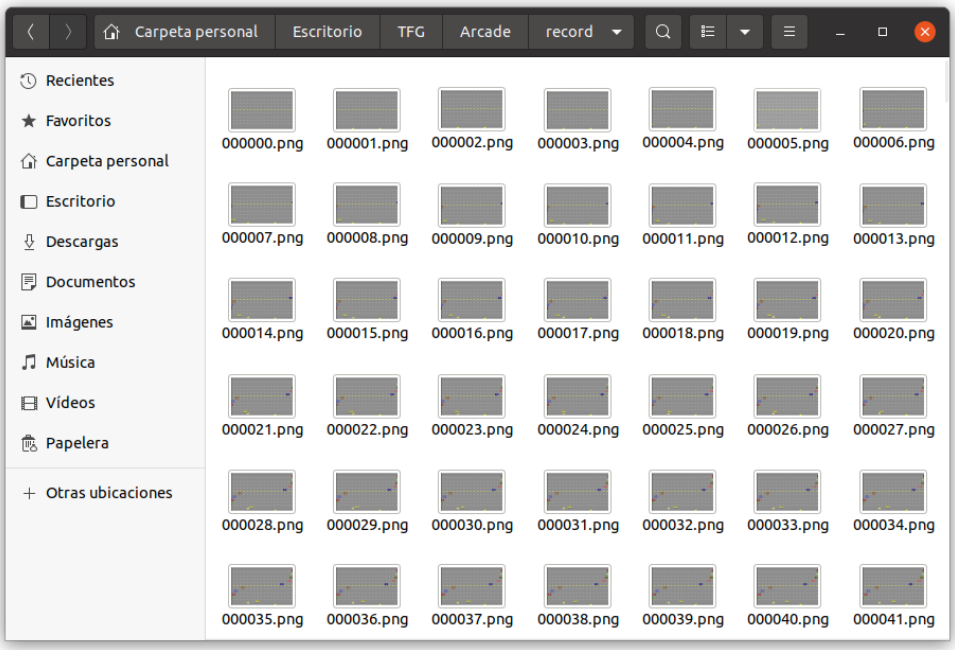

Figura 17: Carpeta "record"

3.4.4 Juegos soportados por la ALE

La Atari 2600 dispone de cientos de juegos y la gran mayoría de ellos están disponibles como ROM para jugar en emuladores. En el siguiente link de *Atarimania*[10] se puede descargar una carpeta comprimida con más de 500 ROMs de juegos de Atari 2600:

http://www.atarimania.com/rom_collection_archive_atari_2600_roms.html

[10] http://www.atarimania.com/index.html

Por desgracia la ALE solo es capaz de soportar unos juegos en concreto, estos dependen de la versión que hayas descargado, pero puedes ver un listado si accedes a la siguiente carpeta de la ALE: `Arcade-Learning-Environment/src/games/supported`

3.5 Freeway

Pare este proyecto se ha elegido un videojuego en concreto con el que trabajar, el juego en cuestión se llama Freeway[7].

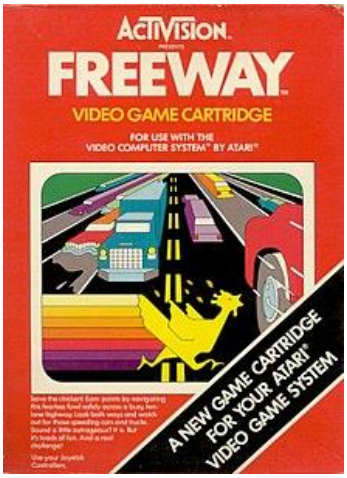

Figura 18: Portada Freeway

Freeway es un juego de la Atari 2600 creado por David Crane y publicado por la empresa de videojuegos Activision[11]. El juego salió a la venta en 1981 y recibió la mención de honor al "juego más innovador" en 1982.

El objetivo del juego es controlar una gallina y cruzar una carretera, el desafío viene dado por los vehículos que transitan la misma. Cada vez que la gallina es atropellada por uno de los vehículos esta retrocede unos metros automáticamente y es alejada de su meta, el lado opuesto de la carretera.

Uno de los motivos por los que este juego es interesante es por su modo de dos jugadores, donde otra persona puede controlar a otra gallina y competir por conseguir más puntos. Cada vez que la gallina llega al lado opuesto de la carretera se obtiene un punto, y el objetivo es conseguir el número máximo de puntos en un tiempo concreto.

[11] https://www.activision.com/

Figura 19: Frame del juego Freeway

Los movimientos de las gallinas están limitados en el eje vertical, haciendo imposible esquivar coches huyendo de ellos, y cada coche tiene su propia velocidad constante, haciendo más complejo su esquive.

4. PROCESAMIENTO DE IMÁGENES

El procesamiento de imágenes es un método para realizar operaciones en una imagen, con el fin de extraer información útil de ella o modificarla. Es un tipo de procesamiento en el que la entrada es una imagen y la salida características extraídas de esta u otra imagen modificada. El procesamiento de imágenes incluye básicamente los tres pasos siguientes:

- Importar una imagen a través de algún tipo de herramienta
- Analizar y/o manipular la imagen importada
- Una salida en la que se puede obtener la imagen alterada o información que se basa en el análisis de la imagen.

Este procesamiento se divide en dos tipos, el procesamiento de imágenes analógico y procesamiento de imágenes digital. Dado nuestro proyecto, utilizaremos técnicas de formato digital; puesto que usaremos las imágenes digitales obtenidas por la ALE tal y como mencionamos en el apartado anterior.

4.1 Imágenes digitales

Una imagen puede ser definida matemáticamente como una función en dos dimensiones, *f(x,y)*, donde *x* e *y* representan coordenadas espaciales (en un plano), y la función f en cualquier combinación de coordenadas, es la intensidad de la imagen en esa coordenada.

Para una imagen digital, cada uno de los elementos con sus coordenadas y su valor es llamado punto elemental o píxel, siendo este la unidad mínima de medida de una imagen digital.

En una imagen RGB, cada píxel se compone por 3 valores que representan la intensidad de cada componente primaria (rojo, verde y azul). El color resultante estará definido por la cantidad de intensidad de cada componente. Si todas las componentes tienen un mismo valor, el píxel se encuentra en escala de grises y será blanco si tienen la máxima intensidad o negro si esta es 0 para todos los componentes.

Figura 20: Circulo cromático RGB

Pero esta no es la única manera de guardar información de un color en un píxel, el formato YUV se aprovecha de la percepción humana asignando 3 valores, uno para la luminosidad y dos para la crominancia[12]. Este es muy útil para compresión dado que el ojo humano es más sensible a la iluminación que a los colores, por ello se puede reducir la calidad de las capas de crominancia reduciendo a su vez la cantidad de información de la imagen, pero prácticamente sin reducir la calidad para nuestra percepción.

4.2 Aplicaciones del procesamiento de imágenes

Como hemos mencionado anteriormente, el procesamiento de imágenes busca la obtención de cierta información de una imagen o la mejora de la misma, en este apartado mencionaremos algunos de usos y explicaremos su funcionamiento desde un punto de vista técnico [8].

Mejora de la imagen
La idea de esta aplicación es simplemente obtener más detalles para mejorar la calidad de la percepción de la imagen destacando características de las que anteriormente carecía o no eran claramente expuestas. Ejemplos de esta aplicación son la eliminación de ruido o los filtros que se utilizan para suavizar la piel de las personas y mostrar menos imperfecciones.

Restauración de una imagen
A diferencia de la mejora de la imagen, esta aplicación busca un resultado más objetivo. La idea es obtener un resultado el cual sea lo más parecido a la imagen original. Esto se lleva a cabo mediante técnicas de restauración tienden a ser modelos probabilísticos o matemáticos para predecir como son ciertas partes con la información de los pixeles restantes. Un ejemplo de esta aplicación es la restauración de fotografías antiguas.

Figura 21: Restauración de una fotografía con procesamiento de imágenes

Procesamiento del color
Procesamientos especiales para modificar la percepción del color. El color es una de las características que puede ser utilizada a la hora de identificar objetos en una imagen.

Compresión
Reduce la cantidad de información de una imagen para reducir su almacenamiento o aumentar la velocidad de envió. Desafortunadamente la calidad de la imagen puede empeorar con este proceso.

[12] Información de los cromas de color

Ondeletas
Otro tipo de compresión de imágenes producida por la transformación de la ondícula para así poder obtener varios grados de resolución.

Procesamiento morfológico
Herramientas para realzar componentes de la imagen, así como la geometría y la forma de los objetos. Un ejemplo es su uso para la creación de máscaras de los objetos de una imagen.

Segmentación
Divide una imagen en regiones. Creando un proceso de clasificación por pixel. Puede ser diferenciado entre: segmentación por color, segmentación semántica, segmentación por texturas o superpíxels.

Representación y descripción
Proceso utilizado para decidir si la forma obtenida es tratada como región o como frontera y extrae fragmentos con información relevante.

4.3 Procesamiento de imágenes aplicado a la ALE

En los puntos posteriores nos centraremos en aquellos procesos utilizados para este proyecto, explicando cuál es su utilidad y su funcionamiento de una manera más extensa. Para ello, debemos imaginar que disponemos de un frame de cualquier juego de la ALE y que mediante el procesamiento de imágenes queremos extraer la información más importante, en nuestro caso la localización de los objetos del juego como coordenadas *x* e *y*.

4.3.1 Background subtraction

Background subtraction (sustracción del fondo) es una técnica para detectar los objetos de una imagen mediante la separación de estos y el fondo. El proceso se hace mediante una sustracción entre los valores de intensidad de la imagen y su fondo o viceversa para todos y cada uno de los píxeles, aquellos píxeles para los que el fondo y la imagen son iguales o similares tendrán una intensidad similar y su resta será 0 o un valor pequeño. En cambio, para aquellos píxeles en los que la imagen y el fondo sean distintos tendrán valores positivos o negativos en función de las intensidades de cada uno, pero como lo que nos interesa es la diferencia de valor entre intensidades, tomamos el valor absoluto de los resultados. Como resultado obtenemos una matriz de dos dimensiones del tamaño de la imagen y si asignamos estos valores cómo intensidades en una nueva imagen, vemos que aquellos píxeles con menor variación de intensidad son más oscuros y aquellos con más variación son más claros a proporcionalmente a la diferencia de intensidad.

Como solo nos interesan los píxeles que conforman los objetos, nuestro objetivo es crear una máscara en la que los píxeles de estos tengan un valor de uno, y los del fondo tengan un valor de 0 (binarizar la imagen). La imagen resultante puede ser binarizada mediante la aplicación de un *threshold*, al que se le asigna un valor que actúa como límite. De esta manera podemos aplicar *background subtraction* en casos donde el fondo varía de intensidad, pero en un rango limitado. Si la diferencia de intensidad es

inferior a ese límite asignamos el valor cero, de lo contrario asignamos un uno. El resultado final es una máscara, qué multiplicada por la imagen original elimina todos aquellos pixeles qué son considerados como fondo [9], tal y como se muestra en la siguiente figura:

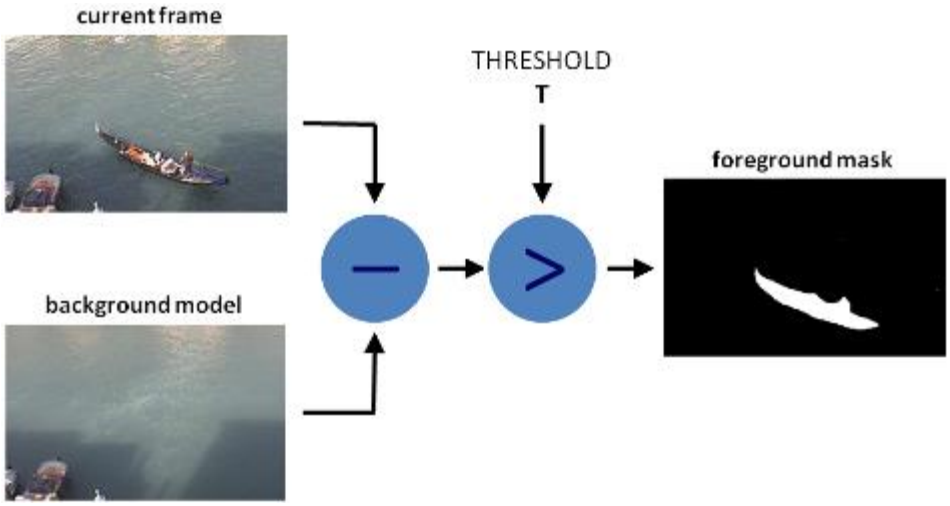

Figura 22: Diagrama de background subtraction

4.3.2 Lukas-Kanade Optical Flow

El método de Lucas-Kanade es un método utilizado para estimar el flujo óptico y detectar el movimiento [10]. Se basa en la idea de que el movimiento de las proximidades de un píxel es similar, es decir, si un píxel forma parte de un objeto que se mueve, aquellos píxeles vecinos que pertenecen al mismo objeto supuestamente se mueven con una velocidad y dirección similar. Con esta idea el método se basa en determinar las variaciones espaciales (movimiento en los ejes *x* e *y*) como las temporales. En nuestro caso, tratamos con imágenes donde la variación temporal hace referencia a los cambios entre frames (vídeo).

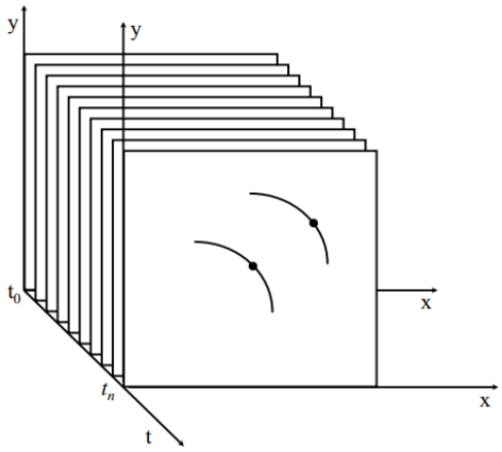

Figura 23: Representación de una secuencia temporal de imágenes[10]

La ecuación del flujo óptico se resuelve por mínimos cuadrados, siguiendo la idea de que, analizando las proximidades de cada píxel de una ventana, estas tendrán un

comportamiento similar. Para ello, se asume que el desplazamiento entre dos instantes próximos temporalmente es pequeño, es decir, se asume que los objetos no se mueven tan deprisa que no hay correlación entre frames. En otras palabras, podemos asumir que la ecuación del flujo óptico debería cumplirse para una ventana centrada alrededor de un píxel en concreto.

Si para una imagen *f(x,y,t)*, donde un píxel de un objeto en la posición *x* e *y* en el instante *t* (frame) se mueve *u* píxeles en el *eje x* y *v* píxeles en el *eje y* para el siguiente frame podemos decir que:

$$f(x,y,t) = f(x+u,y+v,t+1)$$

Ya que la intensidad será la misma, al ser el mismo objeto en otro píxel, y dada la suposición de que los movimientos se producen en desplazamientos espaciales pequeños, podemos utilizar la ecuación de Taylor para desarrollar la siguiente ecuación:

$$f(x,y,t) = f(x,y,t) + u\frac{df}{dx} + v\frac{df}{dy} + 1\frac{df}{dt}$$

De manera que podemos obtener que:

$$u\frac{dI}{dx} + v\frac{dI}{dy} = -\frac{dI}{dt}$$

$$u\,f_x + v\,f_y = -\,f_t$$

Con la ecuación anterior, el siguiente problema es obtener la velocidad ya que disponemos de 2 incógnitas para una sola ecuación. Pero teniendo en cuenta la idea de los mínimos cuadrados de Lukas-Kanade donde se asume que un conjunto de píxeles vecinos, Ω, tendrán el mismo movimiento podemos resolverlo con un sistema de ecuaciones. De manera que podemos expresar la ecuación como:

$$\begin{bmatrix} \sum_\Omega f_x^2 & \sum_\Omega f_x f_y \\ \sum_\Omega f_x f_y & \sum_\Omega f_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum_\Omega f_t f_x \\ \sum_\Omega f_t f_y \end{bmatrix} \equiv A\boldsymbol{v} = b$$

Donde ***v*** representa el flujo óptico del conjunto de píxeles Ω que hay entre dos frames. En la figura posterior se puede observar un ejemplo de implementación del optical flow de Lukas-Kanade en MATLAB[13], donde los vectores rojos indican el flujo óptico de las ventanas (aquellos lugares donde no hay movimiento, no tienen flujo).

[13] https://es.mathworks.com/matlabcentral/fileexchange/48745-lucas-kanade-tutorial-example-2

Figura 24: Implementación del OF de Lukas-Kanade en Matlab[9]

4.4 La salida tras el procesamiento de imágenes

La salida del procesamiento de imágenes será toda aquella información relevante para el aprendizaje reforzado que explicaremos en la sección posterior. La idea para este proyecto es enviar una matriz con las coordenadas de todos y cada uno de los objetos que haya por pantalla. De todos los objetos debemos diferenciar aquel que es controlado por el jugador, en nuestro caso, aquel que será controlado por el agente.

Una vez tenemos los resultados del procesamiento de imagen mencionado anteriormente podemos utilizar la información obtenida por el flujo óptico para encontrar las coordenadas del objeto presenta el jugador, es decir aquel que está controlando mediante las acciones.

Como hemos mencionado en la sección 3.4.1 de características de la ALE, todas las combinaciones de acciones qué puede tomar un jugador están representadas por un valor numérico. Si sabemos qué acción fue tomada para el estado de la ALE del cual hemos obtenido el frame, podemos especular que objeto es aquel controlado por el jugador.

Con el valor la acción y teniendo conocimientos de la tabla podemos saber en que acción fue elegida y por lo tanto que movimiento hace el objeto. Por ejemplo, en el caso de realizar la acción de moverse hacia la derecha, debemos buscar que objetos tienen un flujo óptico indicado por un vector *(u,v)* donde *u* tiene un valor positivo (negativo para el sentido contrario) y *v* sea cero, ya que no se mueve en el *eje y*.

4.5 MATLAB

Como plataforma utilizada para implementar el procesamiento de imágenes explicado anteriormente para este proyecto se ha utilizado MATLAB®[14].

[14] https://es.mathworks.com/products/matlab.html

Figura 25: Logo de MATLAB

MATLAB es un entorno informático cómo plataforma de programación utilizada por ingenieros y científicos para diseñar sistemas, analizar datos u otras aplicaciones como desarrollar algoritmos y crear modelos. Un beneficio de MATLAB es que proporciona conjuntos de herramientas, referidos como *toolbox*, especializadas en diversos ámbitos como el procesamiento de señales, las comunicaciones inalámbricas, el aprendizaje profundo y la inteligencia artificial entre otras. Con su instalación vienen incluidas ciertas librerías para la API de un motor que permite llamar a funciones de MATLAB desde el lenguaje de Python o C++, lo cual nos permite llamar a sus funciones desde el mismo código de la ALE siendo una ventaja para este proyecto.

Uno de los motivos por los que MATLAB fue elegido como plataforma para aplicar el procesamiento de imágenes en este proyecto es una de sus cajas llamada Image Processing Toolbox™ la cual proporciona gran cantidad de algoritmos y aplicaciones relacionadas con el procesamiento, análisis y visualización de imágenes así, como para el desarrollo de nuevos algoritmos. Entre tus aplicaciones destacamos la reducción de ruido, la segmentación de imágenes, la mejora de imágenes y el procesamiento de imágenes tres dimensiones entre otras de manera automatizada.

Muchas de las funciones de esta *toolbox* soportan la generación de código C/C++, pero por desgracia no se ha podido realizar el procesamiento de imágenes de este proyecto con este tipo de generación.

5. APRENDIZAJE REFORZADO

Las personas desde que somos niños aprendemos a base de interactuar con nuestro entorno, es una forma de conexión en la que podemos percibir una clara relación entre causa y efecto. A medida que hemos crecido y hemos ganado más experiencia interactuando con nuestro entorno, obtenemos conocimientos del mismo y somos capaces de predecir las consecuencias de nuestras acciones y saber cuáles nos favorecen o cuales nos perjudican.

5.1 Introducción al aprendizaje reforzado

El aprendizaje reforzado se basa en aprender que hacer en función de la obtención de recompensas del entorno, de manera que aprendamos a obtener la mejor recompensa, es decir, aprender a tomar la decisión de cómo actuar de manera que obtengas el mayor beneficio por ello. La teoría de la siguiente sección está basada en el libro "*Reinforcement Learning: An Introduction"* de Richard S. Sutton y Andrew G. Barto [11]. Un ejemplo sería un niño tocando un cactus por primera vez, el dolor representaría una recompensa negativa y el niño aprendería a partir de esta experiencia, de modo que en el caso de que se vuelva a encontrar en un entorno similar, sabrá que no tocarlo le proporcionará una mayor recompensa evitando el dolor.

Figura 26: Situación base del aprendizaje reforzado

El director de investigación de Google Inc. [12], Peter Norvig dijo:

"No tenemos mejores algoritmos, simplemente tenemos más datos"

Haciendo referencia a que a medida que un agente tiene más experiencia y conocimientos del entorno, éste toma mejores decisiones con el fin de obtener la mayor recompensa. La ventaja del aprendizaje reforzado es su capacidad por conseguir un equilibrio entre la exploración de nuevos escenarios (el estado en el que se encuentre en ese momento) a partir del uso del conocimiento obtenido anteriormente (todos los estados previos) mediante el sistema de recompensas.

5.2 Términos utilizados

Como hemos mencionado el aprendizaje reforzado se basa en 2 componentes, un agente y un entorno.

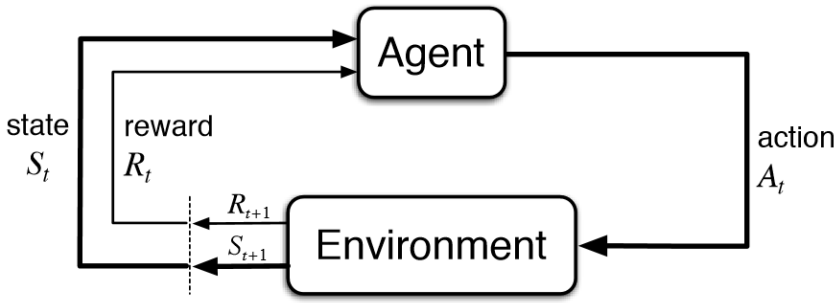

Figura 27: Diagrama del aprendizaje reforzado[11]

Inicialmente el agente se encuentra en un entorno sin recompensa alguna, basándose en la información de dicho entorno, éste toma una acción por consecuencia. Tras esa acción se obtiene un nuevo estado junto con una recompensa del entorno. Con esta información recibida, el agente aprenderá a evaluar la acción que ha tomado anteriormente en función de la recompensa obtenida. Tras este aprendizaje el agente vuelve a tomar otra acción con sus nuevos conocimientos entrando en un bucle hasta llegar a un estado final del entorno.

A continuación, listaremos algunos de los términos utilizados en el aprendizaje reforzado y que serán utilizados en apartados posteriores para una comprensión más técnica.

Términos:

- Agente → El encargado de decidir qué acción tomar a partir de la información que obtiene del entorno.
- Entorno → El mundo en el que actúa el agente y que reacciona a las acciones del mismo con un estado y una recompensa.
- Modelo → Una simulación del entorno para aprender de él en diversas situaciones concretas.
- Estado (s) → Cada una de las situación del entorno anteriores o posteriores a una acción. Cada estado forma parte del conjunto *S.*
- Acción (a) → Movimiento que el agente puede realizar en un entorno. Cada acción forma parte del conjunto *A.*
- Recompensa (r) → Valor generado por el entorno tras una acción, para evaluar el efecto de la misma.
- Política → Determina la manera de actuar del agente, su comportamiento en un entorno. Es una función de estados a acciones que determina la acción por tomar en cada estado.
- Peso (w) → Valores utilizados para calcular que acción tomar a partir de un estado, estos son actualizados con cada estado y recompensa nuevos.

- Valor Q (Q) → Recompensa estimada a partir de un estado y una acción mediante los pesos.

5.3 Tipos de métodos de aprendizaje reforzado

Podemos diferenciar los algoritmos enfocados al aprendizaje reforzado como aquellos que requieren modelos y los que no, así como si siguen una política a la hora de seguir una estrategia.

Algoritmos model-based

En estos algoritmos el agente intenta comprender el entorno y crear un modelo para representarlo y así poder hacer predicciones. Estas predicciones pueden ser aportadas por separado del aprendizaje del agente. De este modo seleccionan la acción anticipándose a los posibles resultados para obtener la mejor recompensa.

Algoritmos model-free

A diferencia de los algoritmos model-based, estos se basan puramente en la experiencia y la información obtenida del entorno y nunca hacen predicciones. Algunos ejemplos son el método de Monte Carlo, SARSA y Q-Learning.

Algoritmos on-policy

Los algoritmos de aprendizaje on-policy son aquellos algoritmos que evalúan y mejoran la política que se utiliza para que el agente seleccione acciones. En otras palabras, intentaremos evaluar y mejorar la misma política que el agente ya está usando para la selección de acciones. SARSA, por ejemplo, es un algoritmo de aprendizaje por refuerzo que estima el valor de la política que sigue. El agente busca la política optima y la usa para actuar a la vez que la actualiza.

Algoritmos off-policy

Los algoritmos de aprendizaje off-policy evalúan y mejoran una política que es distinta de la política que se utiliza para la selección de acciones. Q-Learning es un ejemplo, ya que el agente aprende con la ayuda de una acción obtenida de otra política, en su caso epsilon-greedy.

5.4 El método Monte Carlo

En este apartado hablaremos del método de Monte Carlo para estimar funciones y encontrar políticas óptimas. No nos hace falta tener un conocimiento completo del entorno ya que este método requiere solo experiencia. Dicha experiencia viene dada por los estados, las acciones y las recompensas previas. Incluso sin conocimiento previo a base de aprender con muchas experiencias se puede obtener un comportamiento óptimo. Por otra parte, utilizar modelos también permite al aprendizaje a partir de experiencias simuladas.

El método de Monte Carlo es una forma de resolver el aprendizaje por refuerzo mediante los retornos de la muestra. En nuestro caso lo trataremos como tareas episódicas, ya que nosotros trabajaremos con los estados de la ALE. En otras palabras, consideraremos los estados de la ALE cómo episodios de los que obtendremos la experiencia. Cada vez que termino un episodio, enviarán las estimaciones de valor y las políticas.
Los métodos de Monte Carlo muestran y promedian los retornos para cada estado y acción. Sigue un aprendizaje lineal, es decir, el retorno después de tomar una acción en un estado depende de las acciones tomadas en estados posteriores en el mismo episodio.

Consideramos el método de Monte Carlo para la función del valor esperado, la recompensa futura. Para estimarlo se calcula el promedio de las recompensas obtenidas, a medida que se observan más casos y se obtiene más experiencia este debería converger al valor esperado.

5.5 Finite Markov Decision Process

Es una formalización de la toma de decisiones secuencial, dónde las acciones no necesariamente buscan recompensas inmediatas, si no también situaciones futuras con sus respectivos estados y recompensas. El problema de *Finite Markov Decision Process* (MDP) implica una retroalimentación evaluativa junto con un aspecto asociativo, elegir entre diferentes acciones para diferentes situaciones. Esto implica la existencia de una recompensa mayor a largo plazo y la necesidad de decidir si canjear la recompensa actual o la futura. En lugar de estimar un valor *q(a)* para cada acción *a*, estimamos el valor *q(s,a)* de cada acción *a* en el estado *s*.

En otras palabras, un agente que interactúa con un entorno mediante acciones busca la acción para obtener la mayor recompensa, pero dada la decisión secuencial, una acción influirá en el estado futuro. Por ello, el agente debe tomar decisiones teniendo en cuenta su futuro con la mentalidad de obtener una recompensa superior a las actuales. Por ejemplo, en el caso de saltar si nos encogemos podemos tomar impulso y llegar más alto a pesar de estar retrocediendo en el movimiento inicial, lo cual parece contraproducente.

Como ya hemos mencionado aquel que decide y toma las acciones es el agente, quién interactúa con un entorno el cual responde a la acción, tal y como se muestra en la figura 24. Cada una de estas interacciones representa un nuevo estado, cada estado una representación de la situación del entorno tras una interacción. Inicialmente se empieza con un estado base S_0 para el cual se toma una acción A_0. A partir de este momento se entra en un bucle dónde tras cada acción se obtiene un nuevo estado S_{t+1} con una recompensa R_{t+1}.

Dado que se trata de un sistema secuencial, cada estado y recompensa está influenciado por su estado y acción anteriores. Por ello hay que tener en cuenta la acción y el estado anterior (*s,a*) a la hora de calcular la probabilidad de obtener un estado concreto (*s'*) y una recompensa.

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\},$$

Dado que p representa una probabilidad y el sumatorio de probabilidades de una distribución suma un total de uno podemos afirmar entonces que:

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \,|\, s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s).$$

Si el estado mantiene información de todas las interacciones anteriores con el entorno, entonces todas esas interacciones también repercutirán en el futuro, hablamos de que tiene la propiedad de Markov.

El propósito del agente es obtener la mayor cantidad de recompensas, y las recompensas no son más qué un valor como resultado de una interacción. Un ejemplo para un robot que queremos que se mueva en una dirección sería aportarle una recompensa positiva cada vez que avance hacia el objetivo y darle una recompensa negativa en caso de que se aleje. Dado que el agente busca maximizar la recompensa, el robot terminaría cumpliendo su objetivo, porque es lo que le proporcionaría una recompensa mayor.

Dado que para cada estado se obtiene una recompensa, pero lo que el agente busca no es necesariamente una recompensa inmediata, podemos decir que lo que busca maximizar es el *expected return, G_t*. Por lo general suele ser el sumatorio de recompensas, pero puede ser cualquier tipo de función la secuencia de recompensas.

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

Mediante el *discounting* un agente puede decidir si es mejor recibir una buena recompensa inmediata o tomar acciones que nos lleven a una recompensa mejor en el futuro. Se obtiene mediante la multiplicación del *discounting rate*, representado con la letra *gamma*, elevado a la cantidad de estados necesarios para llegar a esa cierta recompensa.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

La función de valor Q mide cuánta recompensa vamos a recibir si primero tomamos la acción *a* en el estado *s*. Podemos definirla como la esperanza del *expected return, G_t,* para un estado *s* y una acción *a*.

$$Q(s, a) = E[G_t | S_t = s, A_t = a]$$

Para casos en los que la cantidad de estados y acciones es muy grande podemos realizar aproximaciones del valor Q y estimarlo (*estimated Q-value*). Para ello utilizamos un vector de *features x* como mapeado de un estado y acción, junto con un vector de pesos *w*. De este modo, en estados similares y que tengan *features* parecidos se calculará un *estimated Q-value* parecido.

$$\hat{Q}(s, a, w) = f(x(s, a), w)$$

En la expresión anterior x(s,a) hace referencia a los features asociados al estado *s* y la acción *a*.

5.6 Predicciones on-policy con aproximación

En este apartado nos centramos en el estudio de la aproximación de funciones en el aprendizaje reforzado, considerando su uso estimando la función del valor del estado a partir de experiencia obtenida con una política concreta.

La diferencia entre estimar una recompensa y el valor del estado, es que la recompensa viene dada por un único valor por cada estado, mientras que el valor del estado es calculado con una función utilizando los *features* del estado. Esta aproximación no es representada como una tabla, sino como una función parametrizada por un conjunto de pesos *w*.

De esta manera podemos calcular el valor estimado del estado para un estado concreto s dado el vector de pesos. Ajustando los pesos mediante el entrenamiento podemos hacer que generen valor del estado cada vez más acertado al real. Por lo general, asumimos que la cantidad de pesos es menor que la cantidad de estados. Cambiar un solo peso puede alterar el valor estimado varios estados. Esta generalización hace el aprendizaje más potente pero simultáneamente más difícil de manejar y comprender.

5.6.1 Value function aproximation

Hasta ahora hemos definido todas las predicciones cómo actualizaciones de una función para obtener un valor estimado y dichas actualizaciones buscan cambiar el resultado de este valor para acercarse más a un *update target* de un cierto estado.

Nos referimos a una actualización individual con la notación $s \rightarrow u$, dónde *s* representa el estado actualizado y *u* es el *update target* al que el valor estimado de *s* se aproxima.

Podemos interpretar la actualización $s \rightarrow u$ tal que el valor estimado del estado *s* debería ser más parecido al *update target*, *u*. Hasta ahora hablábamos de actualizar el valor estimado para un estado concreto, pero aplicando métodos complejos y sofisticados al implementar la actualización, generalizamos de manera que los valores estimados otros estados también son cambiados.

En *Machine Learning* nos referimos al aprendizaje supervisado aquellos métodos que aprenden a base de observar ejemplos de entradas y salidas. Del mismo modo, el proceso de aproximación de funciones utiliza cada actualización de la predicción del valor estimado como un ejemplo para entrenar.

5.6.2 El objetivo de la predicción

Como hemos mencionado anteriormente, la actualización de un estado concreto afecta a otros y por posible tener valores para todos los estados exactamente correctos, ya que los pesos se habrán variado durante el proceso de la actualización anterior. Para solucionar este problema definimos una distribución μ con la finalidad de representar la importancia del error para cada estado. Nos referimos al error de un estado como la diferencia entre el valor aproximado de cada estado s en un episodio y su valor real

elevada al cuadrado. El *mean square value error* es calculado como el sumatorio del error de cada estado multiplicado por la importancia de ese mismo estado de la distribución μ.

$$\overline{\text{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \Big[v_\pi(s) - \hat{v}(s,\mathbf{w}) \Big]^2$$

Podemos definir la distribución μ como una fracción del tiempo gastado en el estado s. Bajo el entrenamiento con *on-policy*, esto es referido como *on-policy distribution*. El objetivo de *mean square value error* es encontrar un vector de pesos para el cual se genere un valor de *mean square value error* menor que para el de los pesos actuales.

Este objetivo se puede conseguir con aproximación de funciones lineales, pero es más complejo para funciones de redes neuronales.

5.6.3 Métodos de semi-gradiente y gradiente estocástico

Dos métodos de aprendizaje para asignación de funciones en la predicción de valores están basados en el descenso de gradiente estocástico (SGD). Para los métodos del descenso de gradiente, el vector de pesos es representado como una columna de valores reales y la función de aproximación utiliza esos pesos para los estados incluidos en el episodio. Los pesos serán actualizados en cada estado, por lo que dispondremos de un vector de pesos diferente para cada estado w_t.
Por cada estado que observamos tenemos un ejemplo de tu verdadero valor bajo una política concreta. A pesar de tener el valor verdadero, por lo general no podemos obtener un vector de pesos generalizado para todos los estados. Suponemos que los estados aparecen con la misma importancia μ, y tratamos de minimizar el *mean square value error*. Para ello utilizamos el SGD ajustando peso del vector de pesos w en una dirección que reduzca el error.

$$\begin{aligned} \mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha \nabla \Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t) \Big]^2 \\ &= \mathbf{w}_t + \alpha \Big[v_\pi(S_t) - \hat{v}(S_t,\mathbf{w}_t) \Big] \nabla \hat{v}(S_t,\mathbf{w}_t), \end{aligned}$$

La variable α hace referencia a el tamaño del *step*. El gradiente de la función de estimación hace referencia a los *features* correspondientes de cada peso del vector.

El motivo por el que nos interesa utilizar un valor α de *step* pequeño y no uno grande para corregir totalmente el error, es porque no queremos que los pesos no tengan error para un estado concreto, sino que haya un balance.

También podemos encontrar el caso no dispongamos de una función de valor perfecta, sino una versión que por ejemplo contenga ruido, U_t.

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \Big[U_t - \hat{v}(S_t,\mathbf{w}_t) \Big] \nabla \hat{v}(S_t,\mathbf{w}_t).$$

A pesar de este ruido, el método SGD converge a una aproximación óptima del valor real del estado, por lo que a la larga aseguraría una solución óptima.

Los métodos de semi-gradiente (*bootsrapping*), no convergen de manera tan robusta como los de gradiente, pero si en los casos lineales, no solo de una manera más segura, sino que además también más rápida. Otra ventaja es que permiten un aprendizaje continuo, sin necesidad determinar el episodio. Un ejemplo es para el caso de considerar $U_t=R_{t+1}+\gamma\, \tilde{v}(S_{t+1}, w)$ como objetivo.

5.6.4 Métodos lineales

En los casos en los que la función de aproximación una función lineal del vector de pesos, existe un vector valores reales, *x*, de la misma longitud para cada estado *s*.
En los métodos lineales se calcula el valor del estado como el sumatorio de cada peso individual con su respectivo valor real del vector *x*.

$$\hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) \doteq \sum_{i=1}^{d} w_i x_i(s).$$

El vector x es conocido como el vector de *features*[15] de un estado *s* en concreto.

El gradiente de la función de aproximación en la función SDG para un cierto peso, es el *feature* al que este va relacionado, tal que:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[U_t - \hat{v}(S_t,\mathbf{w}_t)\Big]\mathbf{x}(S_t).$$

El SGD linear garantiza una convergencia a un óptimo local posiblemente aproximado a uno global, pero no se puede garantizar. Por ejemplo, el gradiente del algoritmo de Monte Carlo, que converge al óptimo global del *mean square value error* si α disminuye con el tiempo.

El algoritmo del semi-gradiente también converge del mismo modo por la aproximación de función lineal, pero no del mismo modo que el SGD. El vector de pesos también converge, pero no a un óptimo global, si no a un punto cercano.

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t + \alpha\Big(R_{t+1} + \gamma \mathbf{w}_t^\top \mathbf{x}_{t+1} - \mathbf{w}_t^\top \mathbf{x}_t\Big)\mathbf{x}_t \\ &= \mathbf{w}_t + \alpha\Big(R_{t+1}\mathbf{x}_t - \mathbf{x}_t\big(\mathbf{x}_t - \gamma \mathbf{x}_{t+1}\big)^\top \mathbf{w}_t\Big)\end{aligned}$$

5.6.5 Epsilon-greedy

A la hora de tomar decisiones hemos definido que el agente busca obtener la mayor recompensa, ya sea a corto o largo plazo. Si se decide la acción en función de la estimación del valor Q a partir de unos pesos del vector w, siempre se actuará del mismo modo para estados parecidos. El objetivo de *epsilon-greedy* es que el agente para un estado nuevas acciones y así pueda aprender de esas situaciones. La variable épsilon

[15] En nuestro proyecto este vector de *features* es el resultado obtenido de la sección de procesamiento de imágenes.

(ε) hace referencia a la probabilidad de tomar una acción aleatoria a la hora de que el agente se disponga a interactuar.

5.7 Semi-gradiente n-step SARSA

SARSA (State-Action-Reward-State-Action) es un tipo de algoritmo de aprendizaje reforzado que utiliza *Markov Decision Process* para ajustar el valor de la función de estimación basándose en el siguiente estado. Es un algoritmo similar al de Q-Learning pero con un estado y una acción extra.

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha \left[G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}) \right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1})$$

Podemos obtener una versión semi-gradiente de SARSA utilizando un n-step, para generalizar el método de Monte Carlo con el *one-step temporal difference*, si utilizamos un n-step como objetivo para la aproximación.

En él, al principio de un episodio obtenemos un estado inicial y seleccionamos una acción inicial siguiendo la política o *epsilon-greedy*. Entonces entramos en un bucle donde realizamos la acción y a cambio obtenemos una recompensa y un estado nuevos. En nuestro caso obtendríamos el estado en forma de la imagen del frame, y lo convertiríamos en un vector de *features*. De nuevo aplicamos *epsilon-greedy* para decidir si seleccionamos la nueva acción mediante una exploración seleccionando una acción al azar o calculamos el valor estimado para cada acción y seleccionamos aquella que sea superior. Finalmente actualizamos los pesos utilizados para calcular el valor Q.

6. IMPLEMENTACIÓN

Tal y como se ha mencionado en el apartado de introducción, la idea principal de este proyecto es generar un agente que juega al Atari 2600 utilizando como features de entrada información simbólica del frame en el estado en el que se encuentra el juego. En esta sección hablaremos de la implementación llevada a cabo para lograr este objetivo y así comprobar si el agente es capaz de obtener buenos resultados observando su sistema de recompensas.

6.1 Implementación de la ALE

Como hemos mencionado anteriormente, Arcade Learning Environment es la estructura utilizada como base para este proyecto. Utilizaremos la ALE para emular el juego y obtener los frames a los cuales aplicaremos el procesamiento de imágenes. La ALE también será utilizada como base para montar nuestra estructura de aprendizaje reforzado.

Siguiendo los pasos indicados en el apartado 3.4.2 podemos descargar e instalar la ALE desde un sistema operativo Linux. Entre ellos se encuentra la opción de crear proyectos de ejemplo para comprobar su correcto funcionamiento y ejecución de ROMs.

6.2 Obtención de los features

Una vez tenemos acceso a las imágenes de los frames de la ALE, entramos en el apartado de procesamiento de imágenes. El objetivo es obtener información simbólica de cada frame para enviarla en forma de array como *features* para el agente. Desde el punto de vista del agente, esta es toda la información que recibirá del entorno, y por ello es extremadamente importante enviar la información más relevante.

Para empezar, trabajaremos con todas las imágenes en escala de grises. De este modo, cada imagen contiene la información de un solo valor de intensidad por cada pixel. Las imágenes en RGB requieren de tres capas para representar los colores, una información que no aporta nada de valor informativo al agente y hace necesario el uso de matrices de tres dimensiones para su almacenaje.

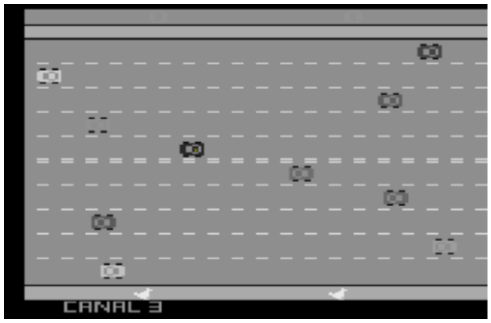

Figura 28: Frame de Freeway en escala de grises

El contenido que nos interesa son los objetos que se observan en cada frame. La información de los píxeles del fondo es completamente innecesaria, siguiendo la teoría del apartado de *background subtraction*, podemos binarizar la imagen separando el

fondo, qué será representado con un valor de 0, de los objetos, los cuales estarán representados con el valor 255.

Para realizar el *background subtraction*, necesitamos a nuestra disposición una imagen del fondo del juego. Para nuestro juego, *Freeway*, no disponemos de ningún frame en el que no se observe ningún objeto por pantalla, por ello debemos generar nosotros mismos la imagen del *background* del escenario de juego.

Para generar el *background*, requerimos de una secuencia de frames en los que todos los objetos deben estar en movimiento. De este modo, si nos fijamos en un píxel concreto y su variación a lo largo del tiempo, veremos el cambio de intensidades que se ha producido en esas coordenadas para todos los frames. Si los objetos se encuentran en movimiento, podemos asumir que están representados por píxeles diferentes ya que habrán cambiado su localización. Con esta información, podemos intuir que el valor de intensidad que se ha repetido más veces en un mismo pixel para toda la secuencia es el del *background*, ya que solo representará a los objetos momentáneamente.

A continuación, mostramos un ejemplo visual de 3 frames diferentes en los que señalamos un mismo pixel.

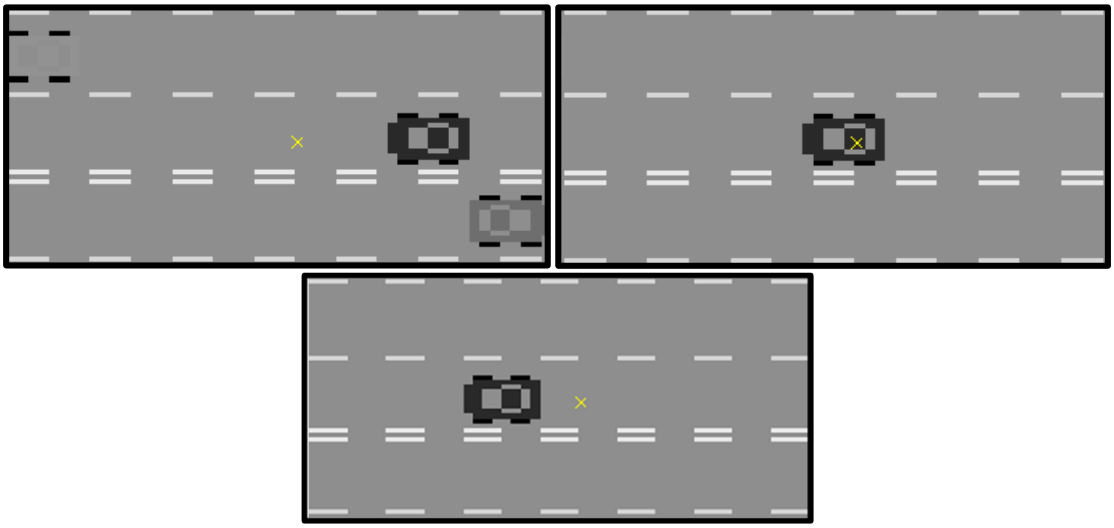

Figura 29: Coordenadas de un mismo píxel en 3 frames

En este ejemplo vemos que el píxel toma dos intensidades distintas, un gris claro para la carretera y uno oscuro para el coche, la intensidad más repetida es la del gris claro así que asumimos esa intensidad como la del *background* en esa localización. Hay que tener en cuenta que a mayor cantidad de frames, mayor probabilidad de que el color más repetido sea el del fondo. Aquellos objetos que no se muevan serán considerados como *background*, tal y como ocurre con el personaje del segundo jugador que puede observarse en la siguiente figura.

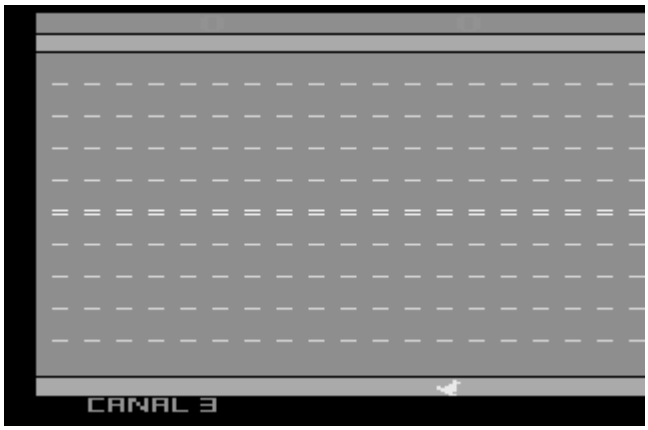

Figura 30: Background del juego Freeway

Una vez disponemos de la imagen del *background* podemos utilizarla para separar los píxeles utilizados para representar objetos en un frame de los que no mediante *background subtraction*. Si realizamos una resta de las matrices con los valores de intensidad de la imagen del frame y la del *background*, aquellos píxeles que tengan valores iguales y estén mismas coordenadas darán como resultado el valor 0, mientras que el resto resultarán en valores positivos o negativos.

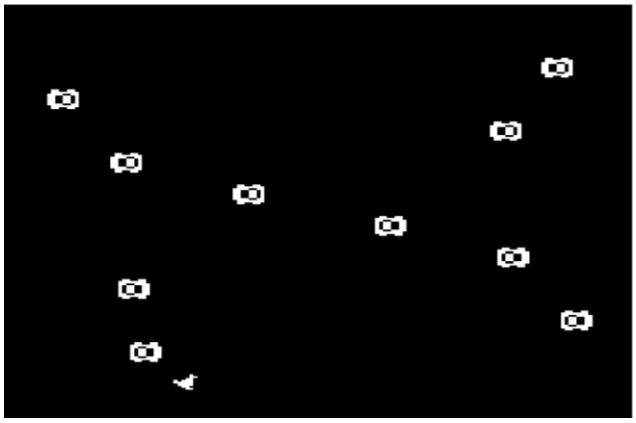
Figura 31: Background substraction de Freeway

Como podemos observar, hemos podido obtener una extracción de los objetos en el frame de *Freeway*. Un detalle a destacar, es que al haber considerado el personaje del segundo jugador como parte del propio *background* este no aparece como objeto. También podemos observar que las ventanas de los coches consideradas como parte del objeto porque tienen el mismo color que la carretera del *background*. Este último detalle se puede solucionar fácilmente, ya que MATLAB dispone de una función qué rellena los agujeros de una imagen binaria a los que no puedo acceder rellenando desde los bordes.

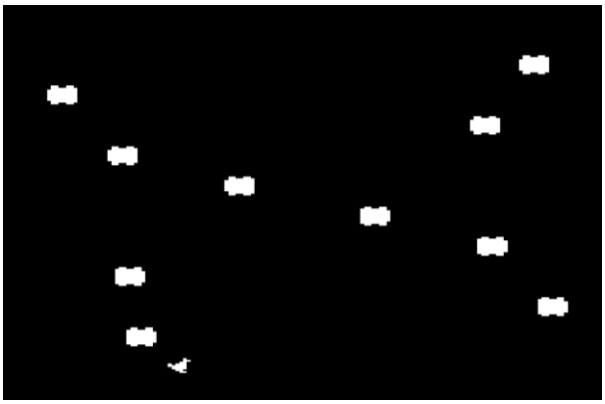

Figura 32: Relleno de la imagen binaria

Con esto hemos reducido drásticamente la cantidad de información, pero podemos reducirla todavía más si almacenamos la localización de los centroides de cada objeto. Para ello utilizamos la función de MATLAB llamada *regionprops*, esta función toma la imagen binarizada y calcula los centroides de sus objetos como la distancia media de todos aquellos píxeles que estén conectados. El resultado es un listado de posiciones de los centros de todos los objetos observados en la pantalla.

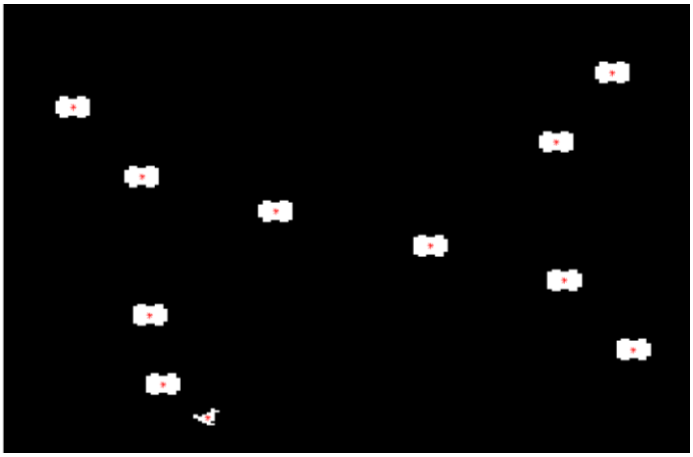

Figura 33: Centroides

Si disponemos de las imágenes de dos frames consecutivos a los que hemos aplicado el procedimiento anterior, podemos obtener el movimiento de sus objetos mediante el *optical flow*. El procedimiento teórico esta explicado en el apartado 4.3.2 de este trabajo, pero resumido, tomamos ventanas para los centroides de los objetos y calculamos la variación espacio temporal de los píxeles que los conforman para estimar el flujo de movimiento de cada uno de los objetos como un vector (u,v). Como se observa en la siguiente figura, solo aquellos objetos que tengan movimiento entre los frames presentaran un flujo óptico en forma de vector.

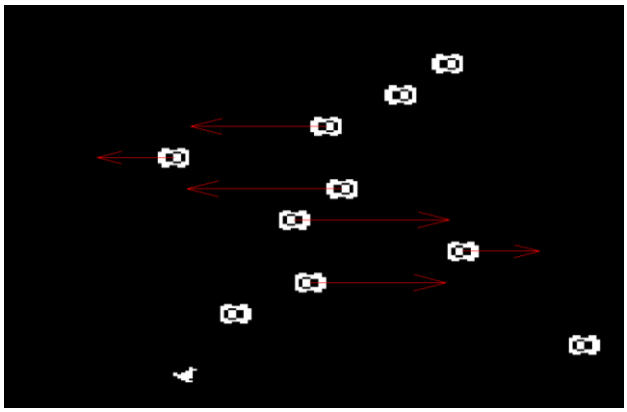

Figura 34: Optical Flow en frames de Freeway

Los frames del juego son el estado original que la ALE nos proporciona cada vez que el agente toma una acción. En el apartado de características de la ALE explicamos que las acciones son representadas con un valor numérico, dado que disponemos de dicho valor, podemos buscar qué objeto es controlado por el agente. El movimiento generado por la acción del agente debe corresponder con los movimientos de alguno de los objetos. En el siguiente ejemplo podemos observar que, tras seleccionar la acción de desplazarse hacia arriba, podemos identificar el personaje por su movimiento ascendente.

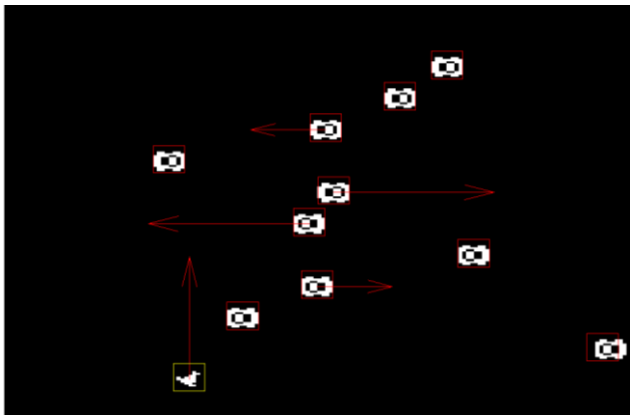

Figura 35: Detección del jugador

Teniendo la localización de los centroides de todos los objetos podemos enviar esta información del frame como los features para el aprendizaje reforzado del agente en forma de matriz de dos dimensiones, con cada centroide siendo representado por un valor para el eje x y otro para el eje y. Si nos fijamos en el juego *Freeway*, nos damos cuenta de que tanto el jugador como el resto de objetos solo se mueven en un eje, así que podemos reducir la cantidad de features eliminando esos valores constantes. Como resultado los features serán in vector con la siguiente disposición:

$$(\mathcal{Y}_0, \mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3, \mathcal{X}_4, \mathcal{X}_5, \mathcal{X}_6, \mathcal{X}_7, \mathcal{X}_8, \mathcal{X}_9, \mathcal{X}_{10})$$

Donde Y_0 hace referencia a la posición del personaje controlado por el jugador en el *eje y* y X_i es la posición en el *eje x* del coche situado en el carril *i*.

6.3 Implementación de SARSA

Para la parte de la implementación del aprendizaje reforzado generamos un código en C++ siguiendo el algoritmo de SARSA n-step explicado en el apartado 5.7 de la sección anterior.

En la siguiente figura podemos observar el pseudocódigo presentado en el libro de "*Reinforcement Learning: An Introduction*" de Richard S. Sutton y Andrew G. Barto [11].

Episodic semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_*$ or q_π

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$, a positive integer n
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
All store and access operations (S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$ or ε-greedy wrt $\hat{q}(S_0, \cdot, \mathbf{w})$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
    If $t < T$, then:
      Take action A_t
      Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
      If S_{t+1} is terminal, then:
        $T \leftarrow t + 1$
      else:
        Select and store $A_{t+1} \sim \pi(\cdot|S_{t+1})$ or ε-greedy wrt $\hat{q}(S_{t+1}, \cdot, \mathbf{w})$
    $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
    If $\tau \geq 0$:
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      If $\tau + n < T$, then $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, \mathbf{w})$ $(G_{\tau:\tau+n})$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[G - \hat{q}(S_\tau, A_\tau, \mathbf{w})\right] \nabla \hat{q}(S_\tau, A_\tau, \mathbf{w})$
  Until $\tau = T - 1$

Figura 36: SARSA's pseudocode

En nuestro caso utilizamos el vector de salida del procesamiento de imágenes como el vector de *features*. Este vector tiene una longitud de 11 variables, pero lo definiremos como un vector de longitud 12, ya que así habrá un valor siempre será uno en el vector al que le acompañará un peso que será independiente para la aproximación lineal. Como el vector de pesos tiene que ser de la misma longitud que el vector de *features* este también tiene longitud 12.

El número de acciones disponible para la ALE es de 17, pero en el juego *Freeway* el jugador solo puede moverse en el eje vertical, es decir que todas las acciones relacionadas con movimientos laterales y de disparo no son necesarias. Y algunas hacen exactamente lo mismo, por ejemplo, la acción de *arriba* y la de *arriba-derecha-disparo* hacen lo mismo, avanzar el personaje del jugador y nada más. Las 3 acciones únicas de este juego que realiza el jugador son: no hacer nada, moverse hacia arriba y moverse hacia abajo.

7. ANÁLISIS DE LOS RESULTADOS

Tras la implementación, nos disponemos a entrenar el agente mediante aprendizaje reforzado. El objetivo de este experimento es demostrar que la gente es capaz de aprender a pesar de recibir poca información, así que si observamos una mejora por lo que respecta a la cantidad de recompensas por episodio, podemos concluir que el agente está mejorando.

El primer problema que sucedió fue dado por los valores de las variables *alpha* y *gamma* a la hora de actualizar los pesos del vector de pesos, w. Dado su tamaño, los valores del vector *w* aumentaban rápidamente. En un principio no supuso un problema, pero a medida que se actualizaba por más y más frames estos seguían aumentando hasta llegar a valores extremadamente grandes, cosa que ralentizaba los cálculos y las actualizaciones haciendo que cada episodio sea más lento de completar que el anterior.

Esto fue solucionado reduciendo el tamaño de ambas variables para generar variaciones menores a la hora de actualizar los pesos.

La primera vez que el agente jugó una partida entera a *Freeway* no obtuvo ninguna recompensa en ninguno de los estados, tras analizar los frames de la partida y la implementación del código vemos problema viene dado una falta de recompensa inicial.

Para entender el motivo debemos entender el funcionamiento del juego *Freeway*. En este juego, el objetivo es cruzar una carretera el máximo de veces posibles durante un periodo de tiempo y cada vez que la carretera es cruzada por el personaje del jugador este obtiene un punto y es teletransportado al lado opuesto (el inicial) de nuevo. La gracia del juego, es que durante el trayecto has de esquivar coches que circulan por dicha carretera. Se obtiene una recompensa por cada vez que se cruza la carretera, pero no existe ningún tipo de recompensa negativa para esta clase de juegos.

Figura 37: Frame de Freeway

La ALE proporciona recompensas negativas en aquellos juegos en los que el jugador dispone de una cierta cantidad de vidas, de manera que cuando las ha perdido todas se termina el juego y por lo tanto el episodio. Cada vez que una de estas vidas es gastada

tras tomar una acción en un estado se proporciona una recompensa negativa para actualizar los pesos.

En casos como el juego de *Freeway*, el jugador no dispone de vidas por lo que no se proporcionan recompensas negativas cada vez que el personaje controlado por el agente es atropellado por uno de los coches. Como resultado el agente no obtenía ninguna recompensa y por lo tanto no tenía ninguna referencia a la hora de actualizar los pesos para saber si estaba haciéndolo bien o no.

Otro problema es que el personaje se quedaba atrapado al inicio de la carretera porque cada vez que un coche lo atropella, este retrocede hacia el inicio y el primer coche con el que se cruza tiene una de las velocidades más rápidas del juego. Por ese motivo era atropellado con mucha frecuencia al inicio y el personaje era retrocedido hasta el inicio.

La solución a dicho problema fue aumentar el valor de *epsilon*, de este modo potenciábamos más la exploración de nuevas acciones en otros estados y había más probabilidades de que el agente cruzase la carretera y obtuviese una recompensa por ello. A partir de estas recompensas, el vector de pesos se actualizará de manera correcta, por lo que podemos reducir el valor de épsilon a medida que pasan lo episodios ya que el agente aprenderá y cada vez será menos necesaria esa exploración en busca de recompensas iniciales.

En un principio el agente fue entrenado por unos pocos episodios para comprobar su correcto funcionamiento.

Por lo general la cantidad de recompensas por episodio era la misma (0,1 o 2 rara vez), aunque en este caso es normal, ya que se ha entrenado durante muy pocos estados y los pesos no estaban lo suficientemente actualizados como para optimizar, pero notamos una mejora respecto al caso anterior donde el agente no era capaz de obtener ninguna recompensa.

Una vez obtiene la primera recompensa para la acción de avanzar a la meta, el vector de pesos se actualiza y el propio agente elige esa acción con más frecuencia. Con tan pocos episodios el agente todavía no está lo suficientemente entrenado como para aprender a evitar los obstáculos (los coches) y colisiona con bastante frecuencia.

En la siguiente gráfica podemos observar la cantidad de recompensas obtenidas por cada episodio tras aprender de los estados previos y la evolución del agente tras 250 episodios. Cada episodio tiene 8150 estados, ya que esa es la cantidad de frames que transcurren en el periodo de tiempo de cada partida en la que el jugador busca obtener el mayor número de recompensas.

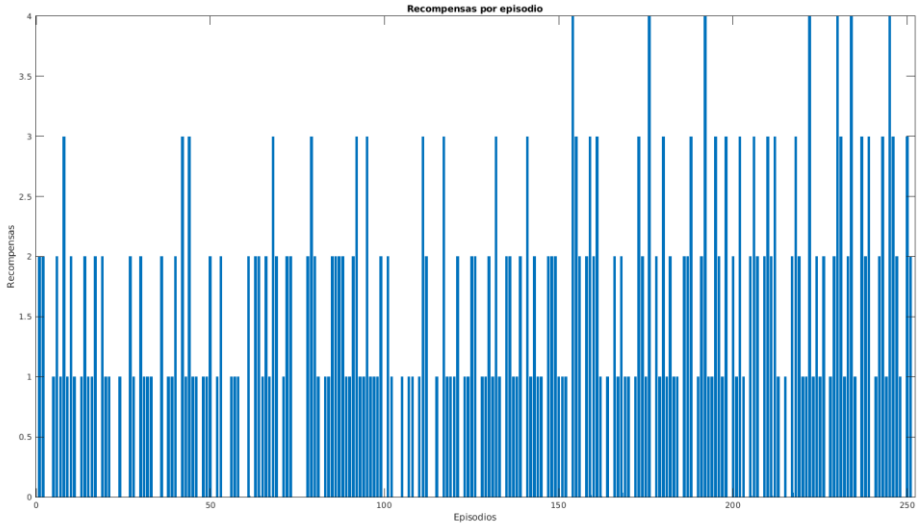

Figura 38: Recompensas por episodio

Como podemos observar, al principio (los 50 primeros episodios) obtiene mayoritariamente entre cero y una recompensa por cada episodio y podemos observar algunos pocos casos de 2 e incluso 3 casos de 3 recompensas en un mismo episodio. En estos primeros estados, el agente apenas estaba entrenado y consideramos gran parte de las recompensas como resultado de la aleatoriedad de la modificación de *epsilon-greedy* para poder obtener las primeras recompensas y actualizar los pesos de esa forma.

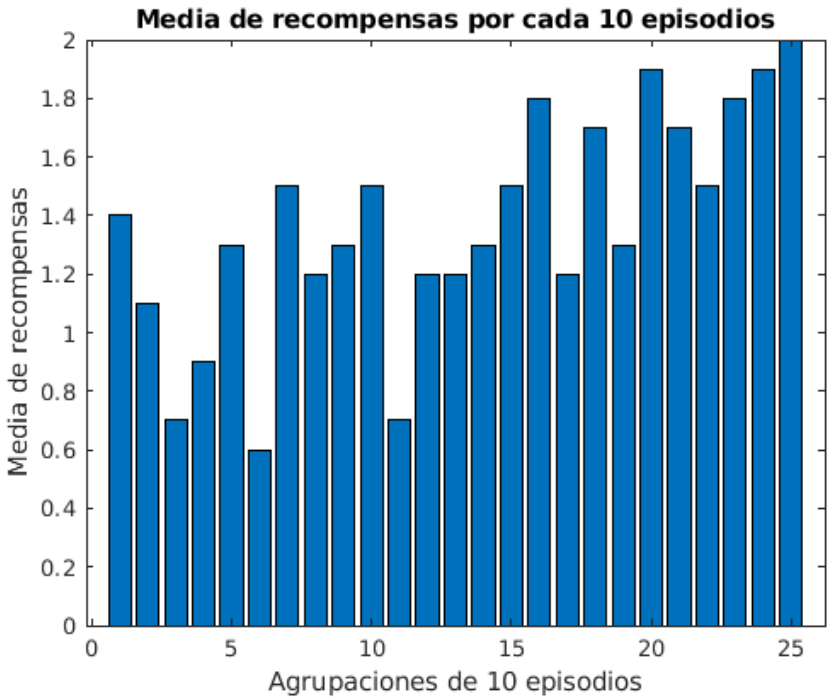

Figura 39: Media de recompensas por cada 10 episodios

Si nos fijamos en la figura anterior, podemos observar la media de la cantidad de recompensas por cada 10 estados, como hemos mencionado anteriormente, atribuimos las primeras recompensas a la exploración de *epsilon-greedy*. Podemos observar que a partir de la 11ª agrupación de 10 episodios la media de las agrupaciones no es inferior a una recompensa, al contrario, vemos como la media aumenta para las medias de las agrupaciones posteriores.

El costo computacional de los entrenamientos es elevado para un ordenador corriente, pero observando los resultados vemos que se produce una clara mejora, podemos suponer que si el agente es entrenado por miles de episodios observaríamos una curva de aprendizaje mucho más clara.

8. CONCLUSIONES

En el siguiente apartado mencionamos las conclusiones extraídas de este proyecto, así como un listado de posibles mejoras aplicables a el mismo.

En primer lugar, hemos demostrado que es posible obtener los datos más relevantes de la imagen de un frame mediante el procesamiento de imágenes, identificar todos los objetos por pantalla e identificar aquel que es controlado por el jugador/agente. En esencia, hemos reducido la cantidad de información de una imagen de 320x210 píxeles (67200 píxeles) a un array de 11 valores para el juego de *Freeway*.

En este proyecto hemos logrado implementar un agente que aprende mediante el aprendizaje reforzado con la estructura ALE como base y el motor de Matlab como intermediario para aplicar el procesamiento de imágenes y obtener los *features* deseados.

Finalmente, hemos demostrado el objetivo principal de este proyecto, que podemos entrenar un agente utilizando como *features* una cantidad de información muy reducida, pero relevante de los frames, y obtener resultados en los que se pueda confirmar su mejora, tal y como ha quedado reflejado por la cantidad de recompensas que se obtiene por cada episodio.

Posibles mejoras para el proyecto:

Entre las mejoras aplicables a este proyecto, destacamos entrenar el agente por muchos más episodios para observar las recompensas obtenidas tras miles de actualizaciones a los pesos, así como obtener una GPU para ejecutar los episodios más rápidamente y entrenar el agente en menos tiempo.

Como ampliaciones al proyecto, se podría utilizar el modo multijugador de *Freeway* para hacer competir dos agentes distintos y así poder compararlos entrenándolos simultáneamente. Recordando el problema de que *Freeway* no dispone de recompensas negativas, otra idea es implementar un sistema de recompensas negativas cada vez que se produce una colisión entre personaje-coche en el juego o aportar una recompensa proporcional a la cercanía de la meta del juego (el lado opuesto de la carretera). También sería interesante comparar los resultados del aprendizaje reforzado sin el procesamiento de imágenes (usando los valores de todos los píxeles como entrada) con los obtenidos en este proyecto.

Utilizando las bases de este proyecto, se puede extraer información simbólica de otros juegos del mismo modo, de manera que podemos observar el agente en otros juegos de la *Atari 2600* y comparar resultados sobre la velocidad de mejora.

Bibliografía

[1] Badia, A. P. (2020, 30 marzo). *Agent57: Outperforming the Atari Human Benchmark.* ArXiv.Org. https://arxiv.org/abs/2003.13350

[2] *Historia de las consolas*. (2012, 14 diciembre). Historia de la Informática. https://histinf.blogs.upv.es/2012/12/14/historia-de-las-consolas/

[3] Rodríguez, A. (2020, 17 noviembre). *La novena generación de consolas de videojuegos empieza a desplegarse en el mercado*. El Comercio. https://www.elcomercio.com/tendencias/novena-generacion-consolas-videojuegos-mercado.html

[4] Alonso, Á. (2014, 30 abril). *Los 20 mejores juegos de Atari 2600*. HobbyConsolas. https://www.hobbyconsolas.com/reportajes/20-mejores-juegos-atari-2600-70100

[5] Bellemare, M. G. (2012, 19 julio). *The Arcade Learning Environment: An Evaluation Platform for General Agents*. ArXiv.Org. https://arxiv.org/abs/1207.4708

[6] Machado, M. C., Hausknecht, M., & Bellemare, M. G. (2018, octubre). Arcade Learning Environment Technical Manual ((v.0.6.0)). Manual.

[7] Wikipedia contributors. (2021, 8 abril). *Freeway (video game).* Wikipedia. https://en.wikipedia.org/wiki/Freeway_(video_game)

[8] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 2nd Edition, Prentice Hall, 2002.

[9] *How to Use Background Subtraction Methods*. (s. f.). OpenCV. Recuperado 4 de mayo de 2021, de https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html

[10] Martinsanz, P. G. (2018). *Desarrollo de una aplicación informática para el análisis de imágenes térmicas - E-Prints Complutense*. Universidad Complutense de Madrid. https://eprints.ucm.es/id/eprint/48850/

[11] Sutton, S. R., & Barto, A. G. (2021). *Reinforcement Learning: An Introduction* (2.a ed.). A Bradford Book. http://incompleteideas.net/book/the-book.html

[12] Weights & Biases. (2020, 20 noviembre). *Peter Norvig, Google's Director of Research – Singularity is in the eye of the beholder* [Vídeo]. YouTube. https://www.youtube.com/watch?v=hVW1mwLtDcI&t=1069s&ab_channel=Weights%26Biases

[13] Sharma, S. (2020, 14 junio). *The Ultimate Beginner's Guide to Reinforcement Learning*. Medium. https://towardsdatascience.com/the-ultimate-beginners-guide-to-reinforcement-learning-588c071af1ec