UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Algoritmos de búsqueda en visor 3D

Documentación técnica

Presentado por Víctor Pérez Esteban
en Universidad de Burgos — 3 de julio de 2017

Tutores: Dr. José Francisco Díez Pastor
Dr. César Ignacio García Osorio

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto *Software*

A.1. Introducción

En esta sección vamos a detallar la planificación que se ha seguido para llevar a cabo el proyecto. Para llevar a cabo la planificación se ha usado ZenHub[1], donde se han ido especificando las tareas realizadas.

La planificación ha seguido la estructura de las metodologías ágiles basadas en iteraciones. Cada iteración consistió de una semana con una reunión con los tutores para analizar el estado del proyecto y las tareas a realizar en esa iteración.

La dirección del repositorio del proyecto es: https://github.com/vpe0001/Algoritmos_de_busqueda_3D-Unity

A.2. Planificación temporal

A continuación vamos a detallar cada iteración, las tareas involucradas y el desarrollo de cada una.

En los cursos previos se realizaron pruebas y prototipos usando versiones anteriores del *software*, tanto de *Unity* como del sistema operativo, pero debido a indisponibilidad temporal no se pudo llevar a cabo. Se utilizo el conocimiento adquirido de estos prototipos sobre los elementos del motor gráfico y sus herramientas para llevar a cabo el proyecto.

[1] Zenhub https://www.zenhub.com/

Iteración 0: Instalación (de 24-04-2017 hasta 1-05-2017)

Las tareas para esta semana fueron preparar el entorno de desarrollo para poder realizar el proyecto. En esta semana tuvimos que probar y solventar los problemas que surgieron con *Unity*, dónde finalmente encontramos una versión anterior que funcionaba correctamente en el equipo de desarrollo.

Las tareas fueron:

- Creación del repositorio.
- Inicio de la memoria.
- Instalación de Unity, TexMaker, SmartGit y ZenHub.
- Iniciar la planificación.

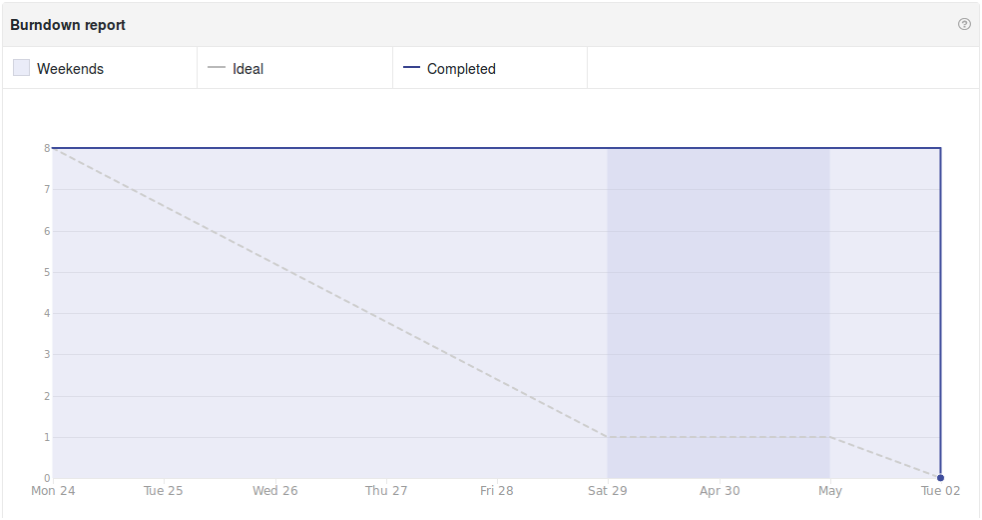

Figura A.1: Gráfico burndown de la iteración 0.

En este caso el gráfico burndown de la figura A.1 muestra que todo el tiempo se fue con los problemas de la instalación de *Unity* y que una vez completada el resto se terminó directamente.

Iteración 1: de 2-05-2017 hasta 7-05-2017

En esta semana se empezó a trabajar con *Unity*, se investigaron los ejemplos de prueba para crear el modelo del vehículo, se creó el primer algoritmo de planificación, el A*, y se investigó cómo se podían representar las rutas obtenidas en Unity llevándose a cabo su implementación.

Las tareas fueron:

- Crear el primer modelo de vehículo con movimiento y físicas de ruedas.
- Crear el A*
- Buscar cómo y realizar el dibujo de trayectorias.

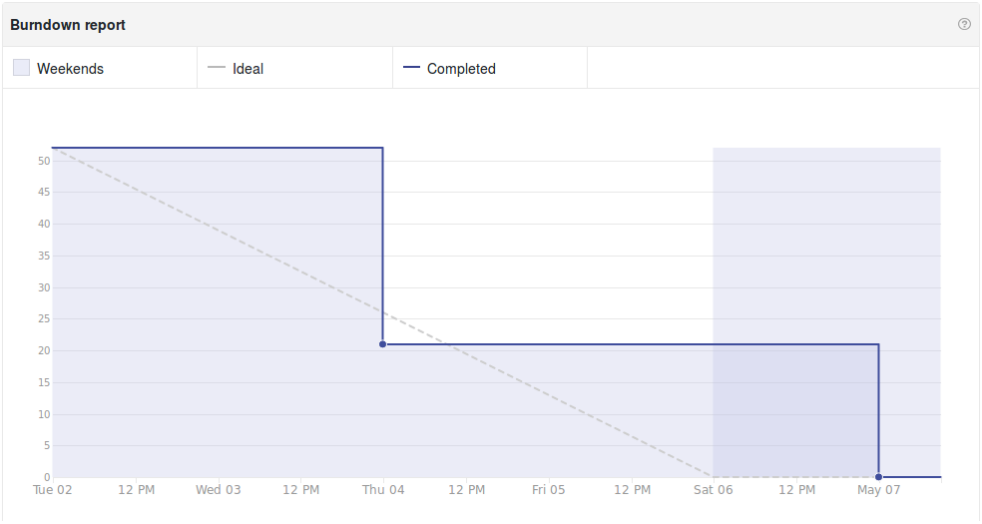

Figura A.2: Gráfico burndown de la iteración 1.

El gráfico burndown de la figura A.2 muestra como se repartió el tiempo entre las dos tareas principales de estudiar los ejemplos de *Unity* y realizar la implementación del A*.

Iteración 2: de 8-05-2017 hasta 15-05-2017

Para esta semana se planificó terminar al A* y la representación de la trayectoria que se inició en la iteración anterior. Además, se investigó y se implementó la representación de los estados del A* para su visualización según se iba ejecutando el algoritmo. Por último, se planificó obtener el mapa de la escena de forma dinámica en tiempo de ejecución, que no fue posible debido a las limitaciones de la versión de *Unity* utilizada, aunque se usó este trabajo y los conocimientos adquiridos sobre los *Nav Mesh* para usarlos en el proyecto.

Las tareas fueron:

- Terminar A*.
- Terminar la representación de la línea de trayectoria.
- Crear la representación gráfica de la cuadrícula de los nodos de estado del A*.
- Crear el mapa para el A* de forma dinámica según la escena.

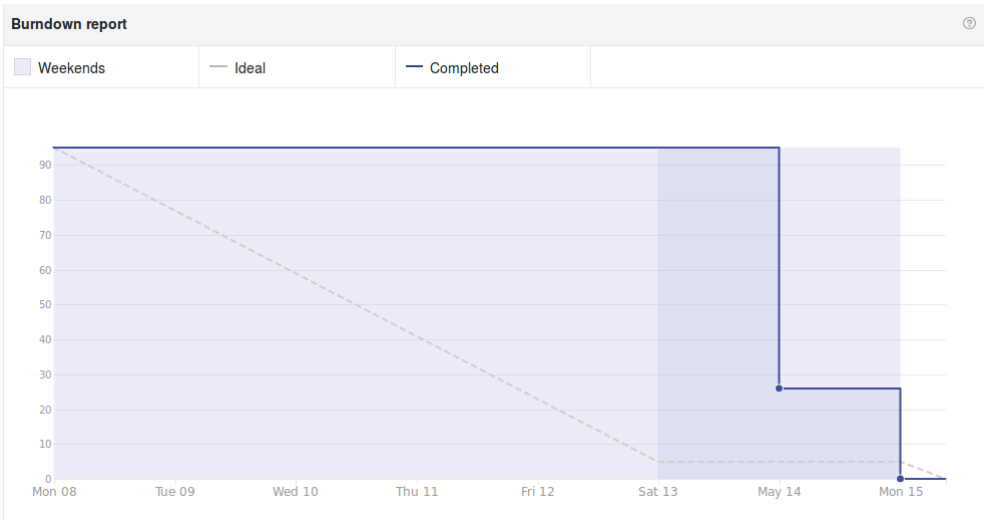

Figura A.3: Gráfico burndown de la iteración 2.

El gráfico burndown de la figura A.3 muestra como la mayor parte del tiempo de esta iteración se utilizó en investigar el *Nav Mesh* y la posible creación de los mapas de forma dinámica.

Iteración 3: de 16-05-2017 hasta 22-05-2017

En esta iteración se implementó el *Path Smoothing* para el suavizado de las rutas obtenidas. Se optimizó el rendimiento del A* a través del cambio de las estructuras de datos utilizadas, en concreto se buscó una implementación de una cola de prioridad. Se avanzó buscando documentación sobre *PID Controller* para la próxima iteración, y se fue realizando la memoria con las secciones desarrolladas hasta ese momento.

Las tareas fueron:

- Implementar el *Path Smoothing* al A*.
- Optimizar el A*.
- Buscar y leer documentación sobre el PID control.
- Avanzar en la memoria.

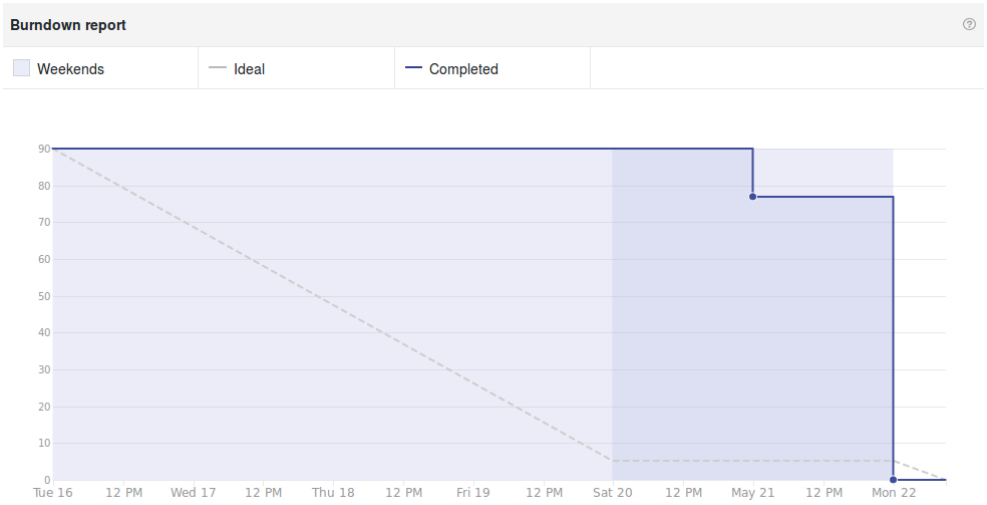

Figura A.4: Gráfico burndown de la iteración 3.

En el gráfico burndown de la figura A.4 podemos observar que gran parte del tiempo de esta iteración se usó en la optimización del A* y en probar diferentes estructuras de datos, y la otra gran parte del tiempo se dedicó a realizar el *Path Smoothing*.

Iteración 4: de 23-05-2017 hasta 30-05-2017

Para esta iteración se utilizó la documentación de la semana anterior para desarrollar el *PID Controller*. Para este semana se planificó realizar un *PID Controller* para el control de las ruedas del vehículo.

También se implementaron dos nuevos algoritmos de búsqueda de rutas, el Theta* y el A* usando los vértices, que mejoraban al A* en la forma de las rutas obtenidas y en el rendimiento en el caso del A* de los vértices.

Se siguió realizando las secciones de la memoria correspondientes al trabajo realizado hasta el momento y se inició la creación de casos de test.

Las tareas fueron:

- Implementar *PID Controller* para las ruedas y el motor.
- Implementar algoritmo Theta*.
- Implementar algoritmo A* con los vértices.
- Avanzar con la memoria.
- Empezar a crear casos de test.

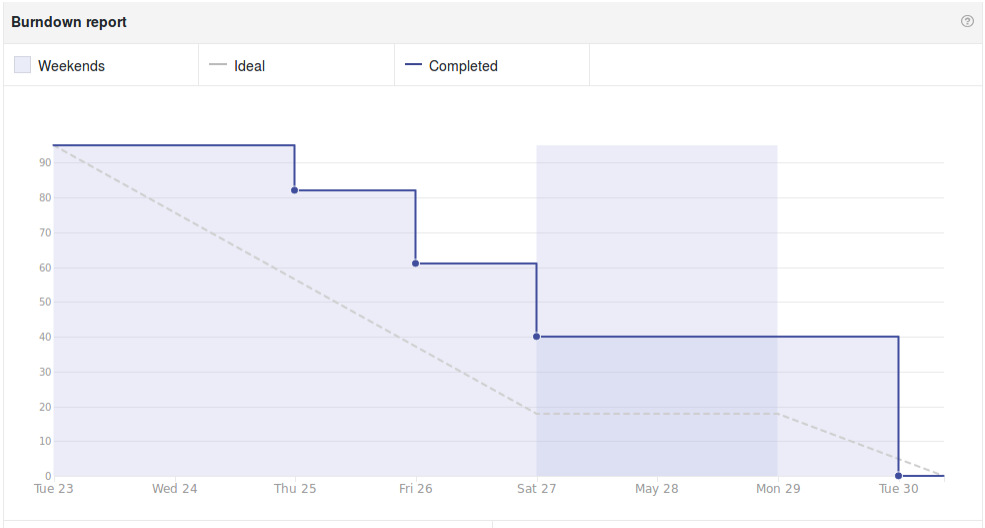

Figura A.5: Gráfico burndown de la iteración 4.

En el gráfico burndown de la figura A.5 podemos ver como la semana se dividió principalmente entre la implementación de los algoritmos basados en al A* y la implementación del PID *Controller*.

Iteración 5: de 31-05-2017 hasta 6-06-2017

Para esta semana estaba planificado realizar un *PID Controller* para el motor del vehículo, pero finalmente se decidió usar solamente el de las ruedas debido a que abarcaba los objetivos del proyecto por si mismo, moviendo el vehículo siguiendo la rutas obtenida, y utilizar en tiempo en otras tareas.

Se dedicó esta iteración a terminar las secciones de la memoria que se habían empezado las semanas anteriores, y a documentarse sobre el algoritmo *Hybrid A** para poder implementarlo en las siguientes iteraciones.

Las tareas fueron:

- Documentarse sobre la algoritmo Hybrid A* para implementarlo.
- Avanzar en la memoria.

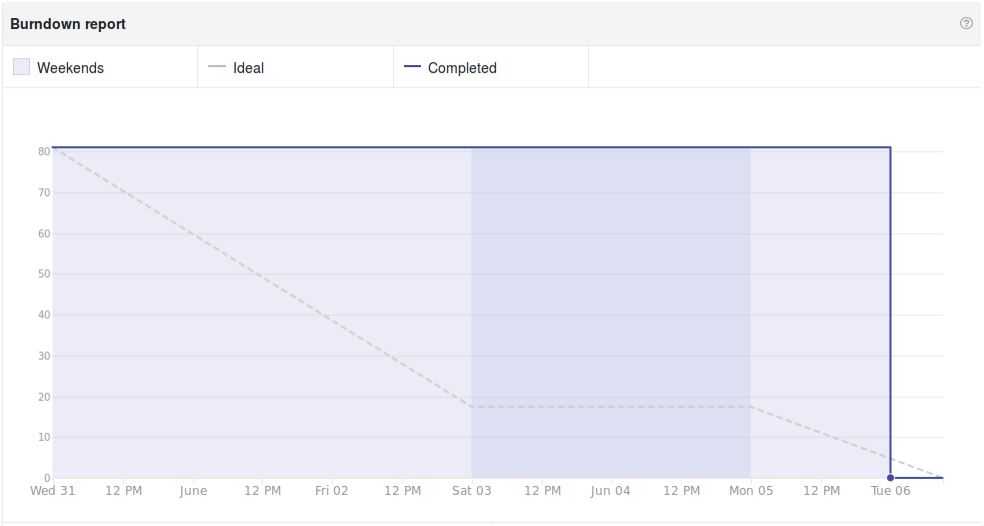

Figura A.6: Gráfico burndown de la iteración 5.

En el gráfico burndown de la A.6 se muestra que el tiempo de esta semana se dedico a buscar documentación sobre el *Hybrid A**.

Iteración 6: de 7-06-2017 hasta 14-06-2017

En esta iteración se implemento una primera versión del *Hybrid A**. Resulto complicado porque aún habiendo realizado buena parte de la documentación durante la semana anterior, no es un algoritmo que esté tan bien documentado y explicado como los anteriores. Finalmente se desarrollo una primera versión donde se calculaba la posición continua del vehículo y tenia en cuenta su rotación, pero resultaban rutas poco realistas y su movimiento a través del *PID Controller* era limitado (por ejemplo, solo era capaz de ir adelante o hacia atrás no, cambiar de sentido).

También se utilizó parte del tiempo para realizar las correcciones a la memoria con los errores encontrados y las indicaciones de los tutores.

Las tareas fueron:

- Implementar el Algoritmo del *Hybrid A**.
- Realizar las correcciones de la memoria.

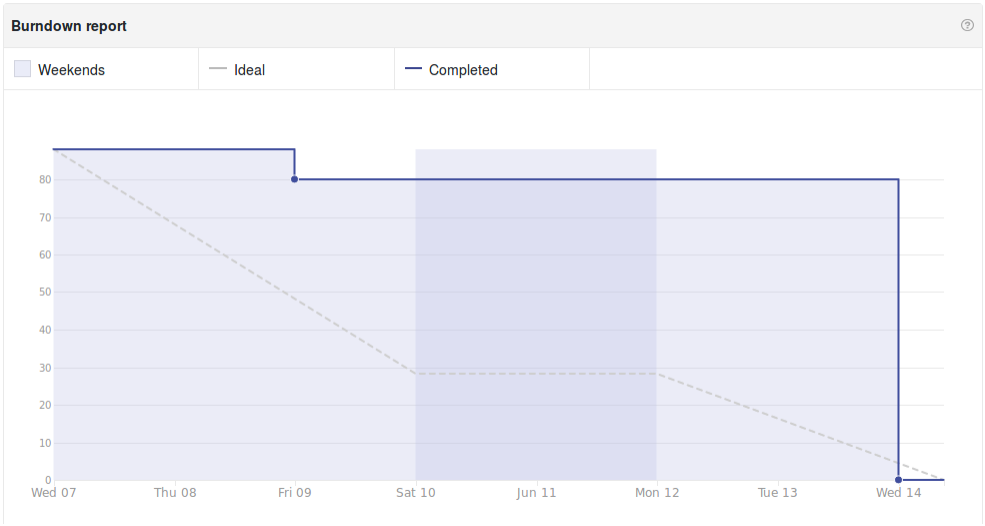

Figura A.7: Gráfico burndown de la iteración 6.

En el gráfico burndown de la figura A.7 se muestra que tuvimos problemas en planificar las subtareas requeridas para le implementación del *Hybrid A** ya que aunque les estudiamos en la semana anterior fue complicado al ser un algoritmo del que no conocíamos nada de antemano y la dificultad de encontrar documentación detallada sobre le mismo.

Iteración 7: de 15-06-2017 hasta 19-06-2017

Esta semana se dedicó a mejorar el *Hybrid A** para que el vehículo pudiera realizar maniobras. Para ello se modificaron las funciones del coste, se arreglaron los errores encontrados en la primera versión y se realizó un mapa de distancias. También fue necesario modificar el *PID Controller* para que fuese capaz de seguir las rutas del *Hybrid A**. De esta manera se consiguió rutas más realistas que pudiera seguir el vehículo y que fuera capaz de realizar maniobras.

Además se siguió realizando la memoria con las secciones correspondientes al trabajo que se había ido realizando hasta el momento.

Las tareas fueron:

- Terminar el algoritmo del *Hybrid A**.
- Avanzar en la memoria.

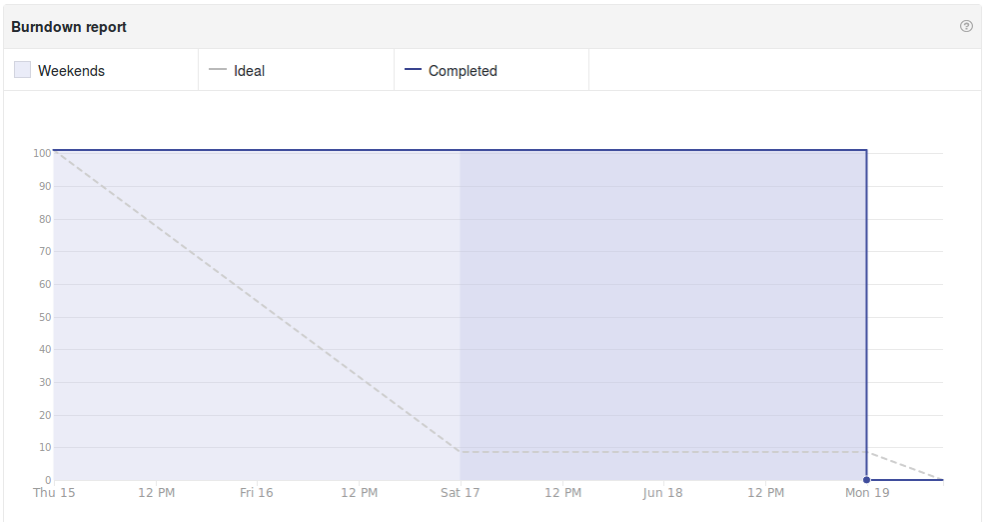

Figura A.8: Gráfico burndown de la iteración 7.

Parecido a la semana anterior, el gráfico burndown de la figura A.8 muestra que una vez terminada la primera versión del *Hybrid A** que no funcionaba como deseábamos, se tuvo que investigar como poder arreglarlo sin tener las subtareas claras al inicio de la planificación.

Iteración 8: de 20-06-2017 hasta 26-06-2017

Para esta semana principalmente se planificó terminar la memoria, añadiendo las últimas secciones, las imágenes, las tablas y los pseudocódigos.

Se finalizó el desarrollo del *Hybrid A** y se mejoró su rendimiento a través de la implementación de una nueva heurística que tiene en cuenta los obstáculos hasta la meta. Se mejoró el cálculo del coste corrigiendo los errores encontrados.

Las tareas fueron:

- Terminar la memoria y realizar las correcciones necesarias.
- Mejorar heurística del *Hybrid A**.
- Corregir errores en el cálculo de las heurísticas y el coste.

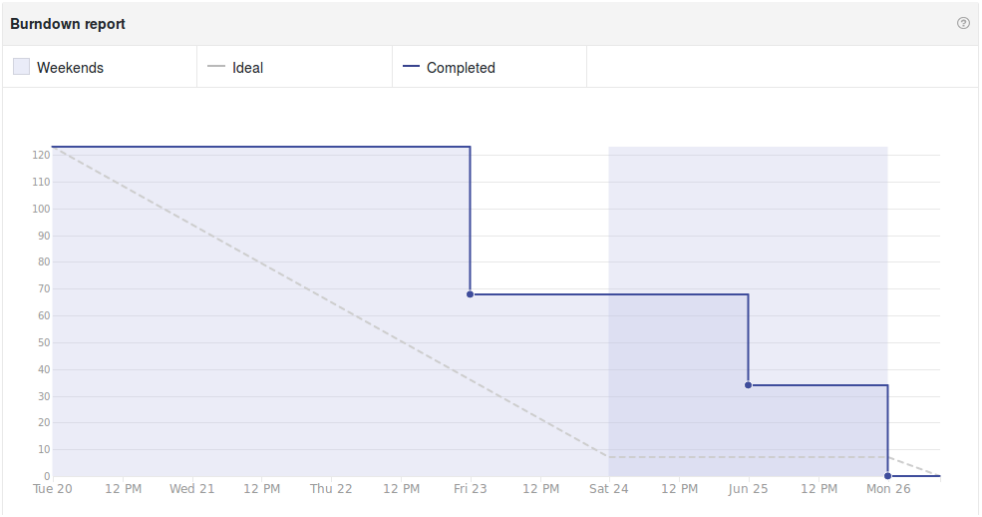

Figura A.9: Gráfico burndown de la iteración 8.

El gráfico burndown de la figura A.9 muestra que en esta semana se planificó mejor las subtareas y el tiempo requerido por las mismas. La mayor parte del tiempo se dedicó a terminar la memoria y el resto a mejorar el *Hybrid A**.

Iteración 9: de 27-06-2017 hasta 3-07-2017

En esta semana se termina el desarrollo del proyecto terminando las tareas pendientes.

Se realizaron los anexos y todas las secciones necesarias, se realizaron los vídeos con los resultados de distintas escenas, los casos de test, y la interfaz de usuario.

Las tareas fueron:

- Terminar los anexos.
- Terminar los tests.
- Implementar la GUI.
- Realizar videos demostrativos.
- Crear el póster de presentación.

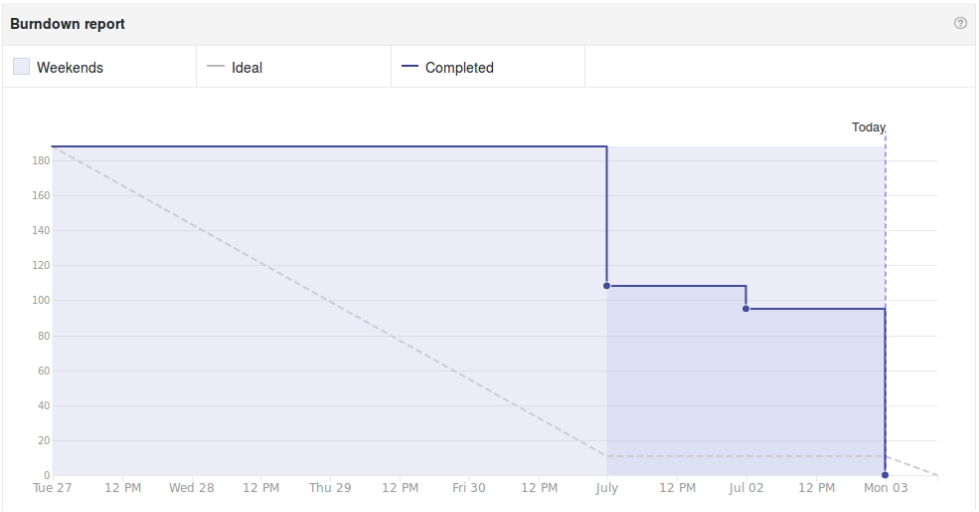

Figura A.10: Gráfico burndown de la iteración 9.

En el gráfico burndown de la figura A.10 se puede observar que algunas tareas de la finalización del proyecto fueron más costosas de lo que se había planeado, ocupando más tiempo del planificado.

Coste	Importe (€)
Salario mensual neto	995.3
IRPF	277.5
Seguridad Social	577.2
Salario Bruto	1850
Total (2 meses)	3700

Tabla A.1: Tabla de costes de personal

A.3. Estudio de viabilidad

Viabilidad económica

En esta sección vamos a analizar el coste económico del proyecto y las posibles vías que pudiera tener para generar ingresos.

Costes de personal

El proyecto se ha llevado a cabo por una persona a tiempo completo durante dos meses. Hemos considerado que el salario neto rondaría los mil euros mensuales, y para calcular el coste total hemos considerado el porcentaje del salario de la Seguridad Social[2] cuyo porcentaje es 31,2 % (contingencias comunes, contrato tiempo parcial y fondo de garantía social) , y del IRPF [3] cuyo porcentaje es 15 % para la elaboración de obras científicas.

En total el gasto por coste de personal para el proyecto es de 3700€.

Costes de *hardware*

Para la realización del proyecto se ha utilizado un PC de sobremesa comprado en el año 2009 por un valor de 1000€ aproximadamente. Vamos a considerar los meses transcurridos desde su compra como el tiempo para su amortización, en total 102 meses.

El coste amortizado de cada mes, por tanto, es de 9.8€, que por dos meses en total el coste del *hardware* ha sido de 19.6€.

[2]Seguridad Social: http://www.seg-social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecotiza36537/index.htm

[3]IRPF: http://www.agenciatributaria.es/static_files/AEAT/Contenidos_Comunes/La_Agencia_Tributaria/Informacion_institucional/Campanias/Retenciones_trabajo_personal/2017/Cuadro_tipo_retencion_2017.pdf

Otros costes	Importe (€)
Internet	50
Impresos y DVDs	60
Alquiler	500
Total (2 meses)	1160

Tabla A.2: Tabla de otros costes

Costes	Importe (€)
Personal	3700
Hardware	19.6
Software	0
Otros	1160
Total	4879.6

Tabla A.3: Costes totales

Costes de *software*

Todo el *software* utilizado ha sido *software* libre o gratuito, incluido el sistema operativo que ha sido Xubuntu 16.04, por lo que el coste del *software* ha sido de 0€.

Otros costes

La realización del proyecto lleva consigo otros costes adicionales que son necesarios para su realización. Hemos considerado el alquiler de un lugar para su realización de un coste aproximado de 500€ mensuales, el coste del proveedor de internet necesario en 50€ mensuales, y el coste de imprimir las memorias, el póster y los DVDs en 60€ .

En total, el coste de estos otros gastos suman un total de 1160€ .

Total costes del proyecto

El total de todos los costes del proyecto asciende a un total de 4879.6€.

Ingresos

En un principio no se ha desarrollado la aplicación para que genere beneficios de ningún tipo, la intención del desarrollo ha sido de investigación y aprendizaje. De todas formas, la forma posible del obtener ingresos y rentabilizar el

proyecto sería a través de la venta en la *Unity Store*[4] de la implementación de los algoritmos como un *asset* para otros proyectos.

Teniendo en cuenta el coste de 4879.6€, con un precio de 30€, y que *Unity* en las condiciones legales[5] para usar su *Store* retiene un 30 %, no recuperaríamos la inversión hasta alcanzar las 233 unidades vendidas.

Viabilidad legal

En esta sección vamos a analizar la viabilidad legal de las licencias usadas en el proyecto así como las licencias con las que se distribuirá.

El motor *Unity*[1] podemos usarlo libremente en su versión Personal siempre que no superemos unos ingresos de 100 mil dolares al año. En cuanto a *Unity* y sus *assets*, se pueden usar libremente dentro del proyecto[6] según los términos legales de la *Unity Store*.

La librería externa que hemos usado ha sido la de la cola de prioridad en C# *High Speed Priority Queue for C Sharp*[7][2] que tiene una licencia de uso MIT [8] que nos permite usarla libremente incluso con fines comerciales.

Con lo cual, la limitaciones que nos encontramos en materia legal serían en caso de comercializar el proyecto, el límite de 100 mil dolares de *Unity*, y tener en cuenta que los *assets* solo se nos permite usarlos como una parte del proyecto, no podemos modificarlos y distribuirlos o venderlos de forma separada.

En cuanto a las licencia del código del proyecto se ha elegido la licencia GPL 3.0, de tal forma que los usos posteriores del código respeten la misma licencia y que deban distribuir el código fuente. Ya que no se ha desarrollado con una intención comercial, si no con intención de investigación y aprendizaje, y debido al uso que hemos dado al código abierto de otras fuentes (programas, ejemplos y la cola de prioridad) para desarrollarlo, consideramos que es la opción más adecuada.

En cuanto a la memoria y la documentación, hemos elegido por los mismo motivos una licencia *Creative Commons* CC BY-NC[9] que permite su uso en cualquier modo siempre que se reconozca la autoría y no se use en obras comerciales.

---

[4] Unity Store: https://store.unity3d.com/

[5] Unity Store, condiciones legales: https://unity3d.com/es/legal/as_provider

[6] Unity Store EULA: https://unity3d.com/es/legal/as_terms

[7] Cola de prioridad: https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp

[8] Licencia cola de prioridad: https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp/blob/master/LICENSE.txt

[9] Licencias Creative Commonshttps://creativecommons.org/licenses/?lang=es_ES

Apéndice B

Especificación de Requisitos

B.1. Introducción

En este anexo se van a especificar cuales son los requisitos que debe cumplir la aplicación una vez finalizada.

La especificación de requisitos sirve tanto para definir el contrato que se estable entre el cliente con el desarrollador, como para la definición formal del análisis de la aplicación.

B.2. Objetivos generales

Los objetivos del proyecto son crear un entorno donde poder desarrollar, estudiar y probar un vehículo autónomo de forma próxima a como se comportaría en la realidad.

En la aplicación final se podrán probar diferentes configuraciones y algoritmos para poder ver y analizar como afectan al comportamiento del vehículo y cuál de ellas es la más apropiada.

B.3. Catálogo de requisitos

Requisitos funcionales

- **RF-1 Simulación del vehículo:** la aplicación tiene que ser capaz de simular el comportamiento de un vehículo de forma realista.
- **RF-2 Configurar escenario:** el usuario debe poder configurar el mapa, colocando obstáculos, modificando la posición del coche y de la meta.
- **RF-3 Elección de los algoritmos:** el usuario debe poder elegir que algoritmo usar y probar con los parámetros necesarios.

- **RF-4 Visualización de los algoritmos:** la aplicación debe mostrar el procedimiento seguido por los algoritmos.
  - **RF-4.1 Visualización del calculo de la ruta:** la aplicación mostrará como se desarrolla la búsqueda de la ruta de forma gráfica.
  - **RF-4.2 Visualización de la ruta:** la aplicación mostrará la ruta obtenida.
  - **RF-4.3 Visualización de los mapas:** la aplicación mostrará los mapas usados para la búsqueda de la ruta.
- **RF-5 Preparación de la ruta:** la aplicación será capaz de procesar la ruta obtenida para adecuarla a la realidad.
  - **RF-5.1 Obtener ruta sin suavizar:** el usuario podrá elegir la ruta sin suavizar tal como la obtenga el algoritmo seleccionado.
  - **RF-5.2 Obtener ruta suavizada:** el usuario podrá elegir la ruta suavizada seleccionando el modo.
- **RF-6 Seguimiento de la ruta:** la aplicación será capaz de mover el vehículo siguiendo las rutas obtenidas.
  - **RF-6.1 Realizar seguimiento manual:** el usuario podrá elegir seguir la ruta con el vehículo de forma manual.
  - **RF-6.2 Realizar desplazamiento automático:** el usuario podrá elegir que el vehículo se desplace por la ruta sin tener en cuenta las limitaciones físicas.
  - **RF-6.3 Realizar seguimiento autónomo:** el usuario podrá elegir que el vehículo siga la ruta de forma autónoma de una forma realista.

Requisitos no funcionales

- **RNF-1 La aplicación debe ser amigable:** debe ser fácil de usar y las representaciones de los algoritmos fáciles de entender.
- **RNF-2 La aplicación debe ser escalable:** debe poder añadirse algoritmos y nuevas funcionalidades en el futuro de forma sencilla.

B.4. Especificación de requisitos

En esta sección se especificarán los casos de uso de la aplicación.

En la figura B.1 se muestra el diagrama con los casos de uso de la aplicación.

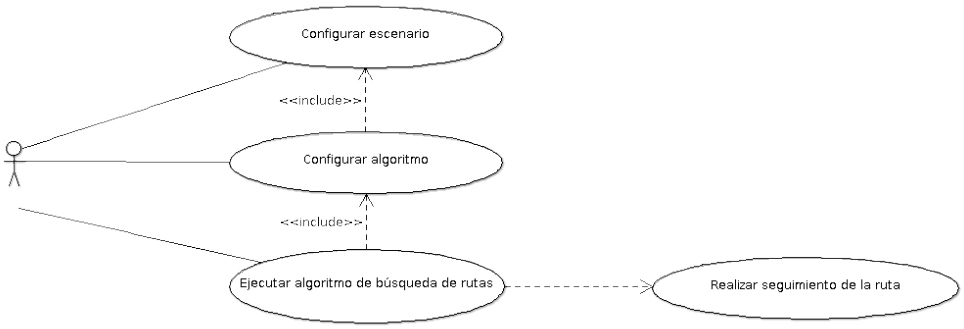

Figura B.1: Diagrama de casos de uso.

CU-01	**Configurar escenario**
Versión	1.0
Requisitos	RF-2
Descripción	El usuario puede modificar el escenario por donde se moverá el vehículo.
Precondición	
Acciones	1. El usuario entra en la aplicación. 2. Se muestran los obstáculos por defecto, el vehículo y la meta. 3. Por cada elemento se muestra los parámetros modificables. 4. El usuario modifica los parámetros de los elementos actuales o añade/elimina obstáculos.
Postcondición	Debe haber al menos la meta y el vehículo.
Importancia	Alta

Tabla B.1: CU-01: Configurar escenario.

CU-02	**Configurar algoritmo**
Versión	1.0
Requisitos	RF-3
Descripción	El usuario puede elegir el algoritmo a utilizar y modificar sus parámetros.
Precondición	Que exista un escenario válido.
Acciones	1. El usuario entra en la aplicación. 2. Se muestran el escenario actual. 3. Se muestran los algoritmos disponibles y los parámetros modificables. 4. El usuario elige que algoritmo usar, con que opciones y modifica los parámetros disponibles.
Postcondición	
Importancia	Alta

Tabla B.2: CU-02: Configurar algoritmo.

CU-03	**Ejecutar algoritmo de búsqueda de rutas**
Versión	1.0
Requisitos	RF-3, RF-4, RF-4.1, RF-4.2, RF-4.3, RF-5, RF-5.1, RF-5.2
Descripción	Se ejecutará el algoritmo seleccionado por el usuario con los parámetros correspondiente y se realizará la visualización del mismo.
Precondición	Que exista un escenario válido.

CU-03	**Ejecutar algoritmo de búsqueda de rutas**
Acciones	1. El usuario entra en la aplicación. 2. Se muestra el escenario actual. 3. Se muestran los algoritmos y los parámetros actuales. 4. El usuario pulsa el botón de ejecución. 5. Si están disponibles y seleccionados, se muestran los mapas correspondientes. 6. Si está seleccionado, se muestra el desarrollo del algoritmo. 7. Si está seleccionado, se realiza la preparación de la ruta con el método elegido. 8. Cuando finalice la búsqueda del algoritmo, se muestra la ruta obtenida.
Postcondición	
Importancia	Alta

Tabla B.3: CU-03: Ejecutar algoritmo de búsqueda de rutas.

CU-04	**Realizar seguimiento de la ruta**
Versión	1.0
Requisitos	RF-1, RF-6, RF-6.1, RF-6.2, RF-6.3
Descripción	El vehículo seguirá la ruta obtenida desde la posición inicial del vehículo hasta la meta.
Precondición	Que se haya encontrado una ruta desde la posición inicial del vehículo hasta la meta.

CU-04	**Realizar seguimiento de la ruta**
Acciones	1. El usuario entra en la aplicación. 2. Se muestra el escenario actual. 3. Se muestran los algoritmos y los parámetros actuales. 4. El usuario pulsa el botón de ejecución. 5. Si el usuario ha elegido el control manual, el vehículo se desplazará con las teclas del teclado. 6. Si el usuario ha elegido el desplazamiento automático, el vehículo se trasladará a través de la ruta ignorando las físicas. 7. Si el usuario ha elegido el seguimiento autónomo, el vehículo se moverá siguiendo la ruta por si mismo, teniendo en cuenta las físicas para un comportamiento realista.
Postcondición	
Importancia	Alta

Tabla B.4: CU-04: Realizar seguimiento de la ruta.

Apéndice C

Especificación de diseño

C.1. Introducción

En esta sección se va a explicar como se ha diseñado la aplicación para que realice los objetivos y requisitos de la misma. También se mostrarán los diagramas correspondientes, aunque se han simplificado omitiendo algunos elementos como los atributos o métodos para mejorar su representación.

C.2. Diseño de datos

La aplicación se divide en dos grandes bloques. Por una parte están los algoritmos que planifican la ruta y por otra están las clases que se dedican a realizar tareas con las rutas que devuelven los algoritmos.

La clase MoverCoche es la clase principal y se encarga de seleccionar el algoritmo, recoger la ruta y proporcionarsela en orden a las clases de suavizado, PathSmoothing, y de movimiento, PIDControl.

También se encarga de instanciar e iniciar las clases auxiliares y parámetros necesarios, y es desde MoverCoche donde se sigue el bucle continuo del motor gráfico para realizar las operaciones.

C.3. Diseño procedimental

Unity como motor gráfico forma parte de la aplicación. La forma de usar *Unity* es a través de *scripts* que heredan de la clase *MonoBehaviour* y enlazandoles con algún elemento de la escena que hayamos creado.

Las clases que heredan de *MonoBehaviour* tienen varios métodos que permiten ser usados cuando se inicia la ejecución, y desde ese punto llamar al resto

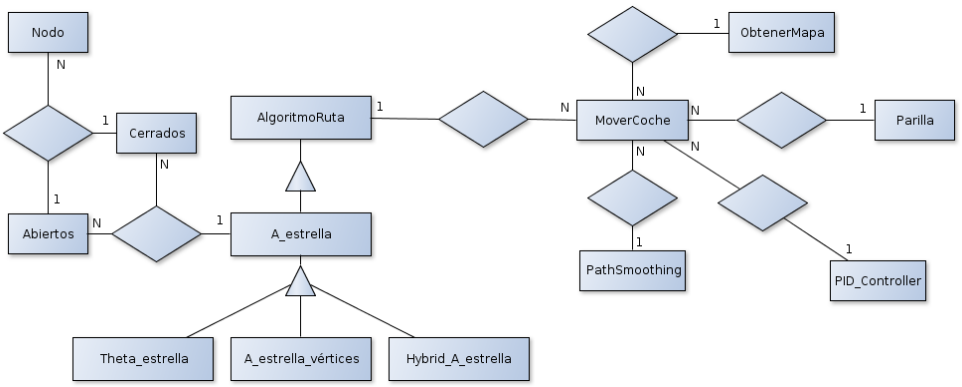

Figura C.1: Diagrama entidad relación.

de clases y métodos. Para este proyecto hemos usado los métodos *Start()*, *FixedUpdate()* y *Update()*.

- ***Start()***: este método se ejecuta cuando se cargue en la escena el objeto al que esta enlazado. Se utiliza para la inicialización de elementos.
- ***FixedUpdate()***: este método se ejecuta a intervalos de tiempo cortos (más rápido que un *frame*), y se utiliza para el cálculo de las físicas.
- ***Update()***: este método se ejecuta cada *frame*, y se utiliza para toda la lógica que no involucre a las físicas.

El vehículo se carga al inicio de la ejecución puesto que está presente en la escena que hemos creado, y por tanto, el primer método que se ejecutará será el *Start()* del *script* MoverCoche que tiene enlazado. Se usa para iniciar todo lo necesario para realizar tanto la ejecución de los algoritmos de búsqueda de rutas, como las clases que hacen uso de la ruta como *PathSmoothing* y *PIDControl*. Es aquí también donde dependiendo de las opciones seleccionados se elegirá un algoritmo u otro para el cálculo de la ruta.

A continuación, el programa se ejecuta como un bucle continuo donde se están constantemente llamando a *Update()* y *FixedUpdate()*. Se ha usado *Update()* para realizar la búsqueda de la ruta, de tal forma que cada llamada a *Update()* corresponde con un ciclo del algoritmo correspondiente. En cada una de esas llamadas también se dibujan el progreso del algoritmo en su búsqueda. *FixedUpdate()* no se utiliza hasta que se ha obtenido una ruta. Al ser usado para el cálculo de las físicas, y por tanto de la simulación del vehículo y su movimiento, hasta que no haya una ruta que seguir no se hace uso de él. Cuando ya se ha obtenido una ruta, se deja de usar *Update()*, y se pasa a mover el vehículo desde *FixedUpdate()*.

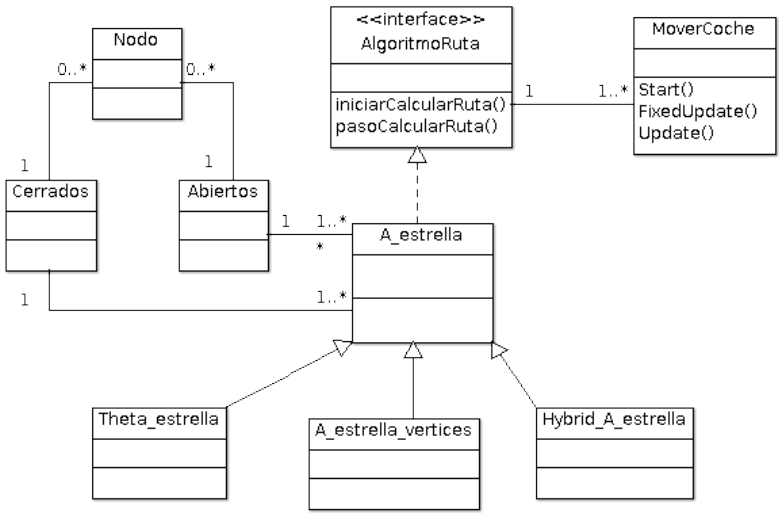

Figura C.2: Diagrama de clases de los algoritmos.

Se ha creado la interfaz AlgoritmoRuta para que en el futuro, implementando los métodos de la interfaz, se pueda instanciar el algoritmo elegido en MoverCoche con lo que facilite la creación de algoritmos nuevos. En el caso de los algoritmos que se han implementado, son tanto el A* como algoritmos que modifican el A*, por lo que hemos usado el A* y se han ido modificando en las subclases los métodos que fueran necesarios.

Una vez obtenida la ruta, se pasa a la siguiente fase de la ejecución. En primer lugar se suaviza la ruta con los métodos de *PathSmoothing*, y a continuación se realiza el movimiento del vehículo. Para el movimiento se realiza desde *FixedUpdate()*, como comentamos anteriormente, y de igual manera se realiza paso a paso en cada una de sus ejecuciones. Para el movimiento autónomo, se llama en cada ejecución a *PIDControl* pata que actualice los valores necesarios para el motor y las ruedas teniendo en cuenta el desplazamiento que ha ocurrido desde la anterior iteración.

C.4. Diseño arquitectónico

En la figura C.4 podemos observar como se relacionan todas las clases principales a partir de MoverCoche.

Las dos primeras llamadas, *iniciarCalcularRuta()* y *getMapaDistancias()*, se ejecutan en el método *Start()* fuera del bucle continuo del motor gráfico.

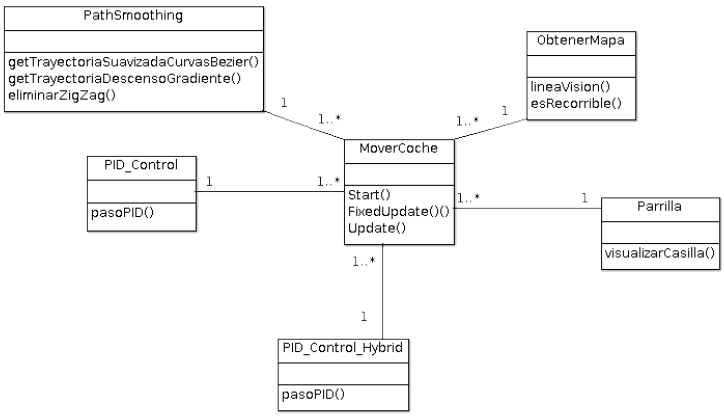

Figura C.3: Diagrama de clases del manejo de las rutas.

Con *iniciarCalcularRuta()* se prepara para iniciarse el algoritmo, creando todos los elementos que necesite. En este ejemplo obtenemos el mapa de distancias puesto que se crea al iniciarse el *Hybrid A** ya que lo necesita durante su ejecución.

A continuación, se realiza la llamada a *pasoCalcularRuta()*, que se ejecuta en *Update()*, formando ya parte del bucle continuo. Al ser *Update()*, se ejecutan una vez por cada frame. Es aquí donde se realiza la búsqueda de la ruta por parte del algoritmo y dibuja el progreso de su ejecución. Cuando termina la búsqueda, en ese ciclo se llama a *getTrayectoriaDescensoGradiente()* para suavizar el resultado.

Y por último, la llamada a *pasoPID()* se realiza en *FixedUpdate()*, que se ejecuta a un intervalo fijo de tiempo, más rápido que a cada *frame*. Como es el método destinado al cálculo de las físicas, se realiza la llamada a *pasoPID()* para obtener los valores de la fuerza del motor y del giro de las ruedas para que el vehículo se mueva de una forma autónoma.

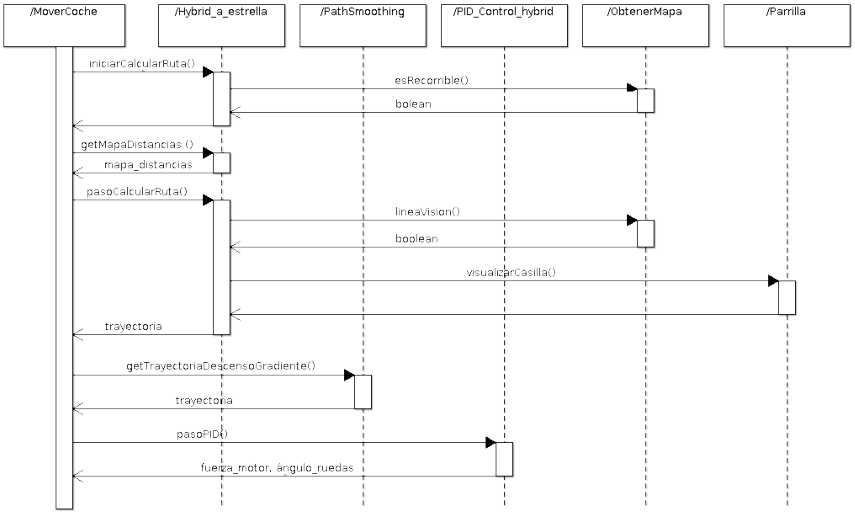

Figura C.4: Diagrama de secuencia usando el algoritmo *Hybrid A**.

Apéndice D

Documentación técnica de programación

D.1. Introducción

En esta sección vamos a detallar como preparar el entorno de desarrollo del proyecto y la estructura que forma.

Hay que tener en cuenta que en parte, también es un manual de usuario, puesto que el proyecto se ha desarrollado pensando en ser ejecutado desde el editor de *Unity* debido a las facilidades que permite a la hora de modificar la escena, los elementos o las pruebas con diferentes parámetros de una forma rápida y directa.

D.2. Estructura de directorios

El proyecto tiene la siguiente estructura de directorios:

- **Build**: es el directorio con los ejecutables.
- **Codigo\Algoritmos_de_busqueda_3D**: es el directorio raíz del proyecto en *Unity*.
- **Codigo\Algoritmos_de_busqueda_3D\Assets**: en este directorio se encuentran todos los elementos que usamos en el proyecto en *Unity*.
  - **Editor\Tests**: contiene los test del proyecto.
  - **Escenas**: contiene las escenas del proyecto.
  - **Materiales**: contiene los materiales creados.
  - **Modelos**: contiene los modelos 3D usados.

    - **Scripts**: contiene los archivos de código fuente.
    - **Texturas**: contiene los texturas usadas.
- **Exportar**: es el directorio con el paquete de *Unity* con el proyecto completo para importarlo a otro proyecto.
- **Memoria**: contiene los archivos en latex con la memoria y los anexos.
- **Vídeos**: contiene vídeos con distintos ejemplos.

A parte de los mencionados, hay más directorios y ficheros creados por *Unity* para su funcionamiento.

D.3. Manual del programador

Instalación de *Unity*

Unity [1] en su versión linux puede instalarse de dos maneras: con el archivo .deb y a través de un instalador en forma de script. Ambos archivos se suministran desde la web oficial y se pueden obtener desde el siguiente enlace:

Web oficial con las versiones de Unity para linux[1]

La versión utilizada ha sido la 5.3.6f1 que se puede encontrar en:

Post de la web oficial para la versión 5.3.6f1[2]

Y descargar de los siguientes enlaces:

- Archivo deb: Archivo deb para versión 5.3.6f1[3]
- Instalador: Instalador para versión 5.3.6f1[4]
- Ambos archivos: Archivo torrent con ambos archivos[5]

Para realizar la instalación hemos usado el archivo instalador con los siguientes pasos:

[1] Web oficial Unity: https://forum.unity3d.com/threads/unity-on-linux-release-notes-and-known-issues.350256/

[2] Unity 5.3.6f1: https://forum.unity3d.com/threads/unity-on-linux-release-notes-and-known-issues.350256/#post-2717623

[3] Unity 5.3.6f1 archivo deb: http://download.unity3d.com/download_unity/linux/unity-editor-5.3.6f1+20160720_amd64.deb

[4] Unity 5.3.6f1 instalador: http://download.unity3d.com/download_unity/linux/unity-editor-installer-5.3.6f1+20160720.sh

[5] Unity 5.3.6f1 ambos archivos: http://files.unity3d.com/levi/unity-editor-5.3.6f1+20160720.torrent

1. Darle permisos de ejecución con chmod +x al archivo instalador
2. Ejecutar el archivo como superusuario. Crea un directorio donde lo ejecutemos donde se encontraran todos los archivos de Unity.

Para ejecutarlo, hay que ir al nuevo directorio que creó el instalador y ejecutar el ejecutable del editor de Unity que se encuentra dentro de *Editor/*.

Para poder usar Monodevelop con la versión de *Unity*, hay que instalar además de la versión que viene con el instalador, el resto de archivos de Monodevelop. Con el comando sudo *apt-get install mono-complete* se instalan desde los repositorios todos los archivos necesarios.

Para comprobar que *Unity* abrirá el Monodevelop con su versión para *Unity*, desde el menú *Edit->Preferences*, en la sección *External Tools*, comprobamos que el editor de los *scripts* seleccionado es *internal*. Aunque se puede usar cualquier editor para realizar los *scripts*, es recomendable usar Monodevelop de esta forma porque hay diferencias entre los lenguajes de programación estándar y las versiones de ellos que usa *Unity*.

También hace falta instalar librerías adicionales de MonoDevelop para poder usar el debugger. Podemos hacerlo desde un terminal con los siguientes comandos:

- *sudo apt-get install mono-reference-assemblies-2.0*
- *sudo apt-get install mono-reference-assemblies-3.5*
- *sudo apt-get install mono-reference-assemblies-4.0*

Para usar el *debugger* de MonoDevelop, hay que iniciar el proceso desde el editor de MonoDevelop, usando el botón de ejecución y las opciones de *Debugger* y *Unity Editor* como se ve en la figura D.1.

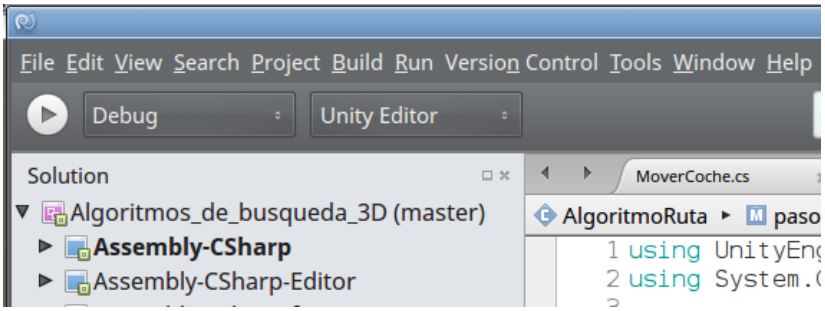

Figura D.1: *Debugger* para *Unity Editor* de MonoDevelop.

Entonces nos preguntará a que proceso de *Unity Editor* queremos enlazarlo si no puede hacerlo el mismo. Seleccionamos el proceso de nuestro proyecto (podemos comprobar cual es con un gestor de procesos si no lo sabemos), y MonoDevelop quedará en espera. El proceso de *debug* no comienza hasta que desde el *Unity Editor* iniciemos la ejecución de forma normal, como lo haríamos sin el *debugger*.

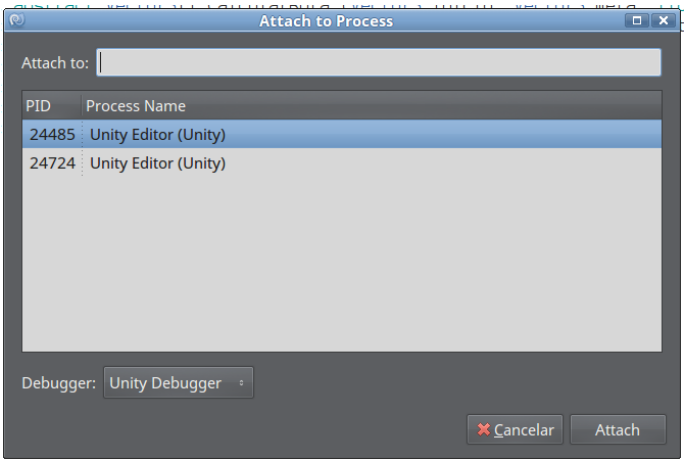

Figura D.2: Ventana donde se pregunta a que proceso de*Unity Editor* enlazaremos el *Debugger*.

Instalación SmartGit

Para la organización del repositorio hemos elegido SmartGit, que es un programa disponible de forma gratuita para uso no comercial con versión para linux.

Para su instalación, hay que descargarse el archivo deb de su página web:

Página de SmartGit[6]

La versión que hemos utilizado es la 17.0.3 que se puede descargar del siguiente enlace:

Archivo deb de SmartGit 17.0.3[7]

El modo de instalación que hemos usado es haciendo doble *click* sobre el archivo deb abriéndolo con el *Instalador de software de Ubuntu*. También se puede instalar con el comando *dpkg -i nombre-archivo.deb*.

[6]SmartGit: https://www.syntevo.com/smartgit/download
[7]SmartGit 17.0.3: https://www.syntevo.com/smartgit/download?file=smartgit/smartgit-17_0_3.deb

Instalación de Texmaker

Texmaker lo hemos instalado desde los repositorios de Ubuntu usando el gestor con interfaz gráfica Synaptic.

También se puede instalar con el comando *apt-get install textmaker*.

Instalación ZenHub

Para la planificación y control del proyecto hemos usado Zenhub sobre github. Es un complemento de Firefox que añade funciones como las *boards* para manejar el control de las *issues* y las *milestones* del proyecto, y que permite ver los gráficos *burndown* del progreso que se ha realizado.

Para instalarlo, hay que ir a la web oficial Web oficial ZenHub[8]

Pulsando sobre el botón de añadir ZenHub a Github, Firefox nos preguntará si queremos permitir a esa web instalar complementos, y debemos darle permiso. A continuación, se instalará en Firefox de forma automática y cuando vayamos a nuestro proyecto en github tendremos disponibles la funciones adicionales.

Instalación de Remarkable

Remarkable lo hemos usando el archivo deb descargado de la página oficial. La versión utilizada ha sido la 1.87, que se puede encontrar aquí:

- Página oficial[9]
- Archivo deb[10]

También se puede instalar con el comando *dpkg -i nombre-archivo.deb*.

D.4. Compilación, instalación y ejecución del proyecto

Compilación

El proceso de compilación de los *scripts* se realiza automáticamente por el editor de *Unity* cada vez que se realiza un cambio en alguno de ellos. Por tanto, no hay ningún comando o manera específica de hacerlo.

[8] Zenhub: https://www.zenhub.com/
[9] Remarkable, página oficial: https://remarkableapp.github.io/
[10] Remarkable, archivo deb: https://remarkableapp.github.io/files/remarkable_1.87_all.deb

Para crear un archivo ejecutable con el proyecto, que irá acompañado de un directorio con los modelos y elementos necesarios para su ejecución, desde el editor de Unity vamos a *File->Build Settings*.

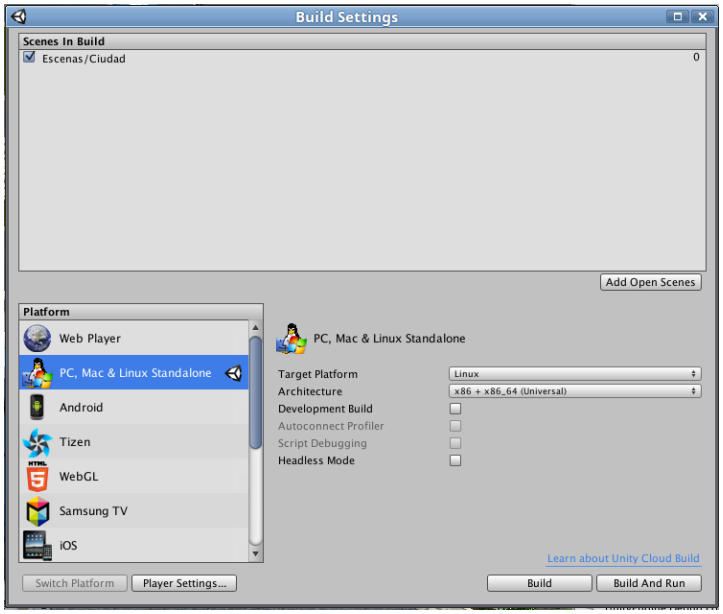

Figura D.3: Ventana con las opciones de creación del ejecutable.

Desde el menú de *Build Settings*, elegimos las escenas que queramos incluir en el ejecutable, la plataforma en donde se ejecutará y el tipo de ejecutable que deseemos crear. A continuación, pulsamos en *Build* y nos pedirá un nombre para el archivo y una ruta en donde se creará. Uan vez introducidos, comenzará el proceso de creación del archivo ejecutable.

Instalación

Para instalar el proyecto, el repositorio de Github con todos los ficheros necesarios es: Repositorio del proyecto[11]

Con git o a través de la descarga del archivo zip, creamos una directorio con los archivos del proyecto. Desde *Unity*, podemos abrir el proyecto que se encuentra en el directorio *Codigo\Algoritmos_de_busqueda_3D*.

También se puede instalar desde *Unity* después de crear un proyecto nuevo, desde el menú , importando el paquete de *Unity* del directorio *Exportar*.

[11] Repositorio del proyecto: https://github.com/vpe0001/Algoritmos_de_busqueda_3D-Unity

Ejecución

Para ejecutarlo desde *Unity*, hay que pulsar el botón de la flecha como podemos ver en la figura D.4. Para parar la ejecución hay que volver a pulsar el botón de la flecha, y es posible pausarla con el boton de las dos barras verticales.

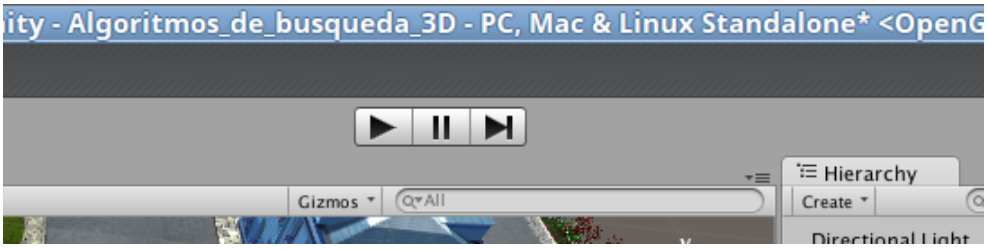

Figura D.4: Botones de ejecución en *Unity*.

D.5. Pruebas del sistema

Para la realización de los tests hemos usado la herramienta del editor de *Unity*, que usa el *framework* de NUnit. *Unity* siempre va a buscar los archivos de tests en el directorio *Editor*, y por ese motivo hemos colocado un directorio llamado *Tests* dentro del mismo.

Para ejecutar los tests, desde el editor de *Unity*, abrimos el *Editor Test Runner* desde el menú *Window*, y pulsamos sobre *Run All* para realizarlos todos, o *Run Selected* para ejecutar los tests concretos que se desee.

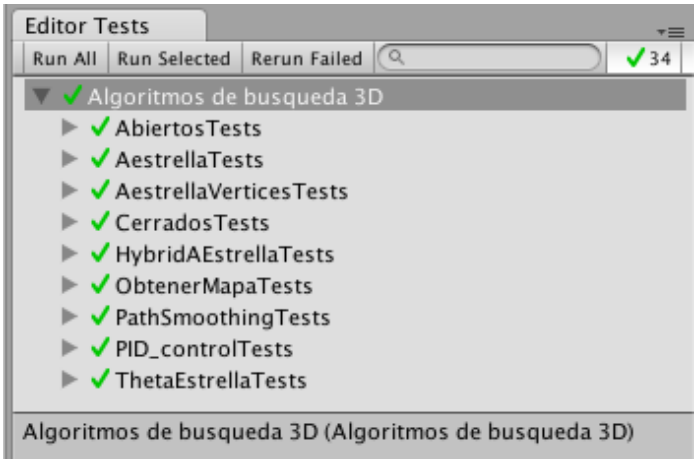

Figura D.5: Ejecución de tests en *Unity*.

Se han implementado casos de prueba para la mayoría de las clases, aunque por ejemplo, la clase MoverCoche no permite la realización de test debido a que al heredar de *MonoBehaviour*, solo el Editor de *Unity* puede instanciarla.

Para los tests de los algoritmos se ha realizado una búsqueda de ruta tanto si era posible alcanzar el destino, como si no lo era, dando lugar al error correspondiente.

Para las clases de los conjuntos de datos se ha comprobado que cada método realiza correctamente las operaciones de manipulación de los elementos que almacenan.

En el caso del *Hybrid A**, además de comprobar las rutas, se ha comprobado que los mapas que genera de obstáculos y de distancias son correctos.

En el PID_Control se ha comprobado que los ángulos que calcula respecto a las posiciones del vehículo son correctos, y que los valores para un momento conocido para el movimiento del vehículo son los esperados.

En el *PathSmoothing*, se ha comprobado que el descenso gradiente se obtienen los valores correctos al compararlos con los proporcionados por la documentación que se siguió para su implementación. Se comprobó que la eliminación del zigzag eliminaba estados que no eran necesarios, y que las curvas Bézier generan los puntos correctos y que sus valores eran los esperados.

En ObtenerMapa se comprobó que detectaba correctamente cuando una posición correspondía a un obstáculo y cuando no, y también se hicieron pruebas con los resultados obtenidos con la línea de visión para comprobar que el resultado era correcto.

Apéndice E

Documentación de usuario

E.1. Introducción

En esta sección se va detallar como utilizar la aplicación. Como usuarios hay dos posibilidades. La versión ejecutable, que permite ver una serie de escenarios de prueba, y usar la aplicación directamente desde *Unity*, lo que permite poder modificar las escenas, los parámetros y contar con una vista adicional para poder seguir el desarrollo de los algoritmos.

E.2. Requisitos de usuarios

Para poder ejecutar el proyecto desde *Unity*, es necesario la versión 5.3.6 o superior, y un ordenador con cualquiera de los sistemas operativos principales (Linux, Windows y MacOS) que cuente con una tarjeta gráfica 3D.

Para el ejecutable los requisitos son los mismos solo que no es necesario usar *Unity*.

Los requisitos mínimos oficiales para usar *Unity* son:[1]

- Sistema operativo Windows 7 SP1 o superior, MacOS X 10.8 o superior, Ubuntu 12.04 o superior.
- Tarjeta gráfica con DX9 (modelo de shader 3.0) o DX11 con capacidades de funciones de nivel 9.3.
- CPU compatible con el conjunto de instrucciones SSE2.

[1] Requisitos mínimos Unity: https://unity3d.com/es/unity/system-requirements

E.3. Instalación

El proceso de instalación para usar la aplicación con el editor de *Unity* es la misma que vimos en la sección D.4.

Todos los archivos necesarios están en el repositorio de github, cuya dirección es: Repositorio del proyecto[2]

Para descargarlo, se puede usar git o descargar el archivo zip y descomprimir los ficheros. Desde *Unity*, se abre el proyecto que se encuentra en la carpeta *Codigo\Algoritmos_de_busqueda_3D*, o bien se crea un proyecto nuevo y se importa el paquete del directorio *Exportar*, desde *Assets->Import Package*.

El ejecutable no requiere de instalación adicional. Los archivos ejecutables se encuentran en la carpeta*Build*, en la que a su vez hay dos carpetas, una para la versión Linux, y otra para la versión Windows.

E.4. Manual del usuario

Manual para *Unity*

Una vez realizada la instalación, abriremos el proyecto con *Unity*. Cuando se inicie, nos encontremos en la interfaz del editor de *Unity*. Desde *File->Open Scene* podemos cargar la escena que deseemos probar. Nos encontraremos una pantalla similar a la de la figura E.1.

A la izquierda, tenemos dos vistas de la escena en tres dimensiones. La de arriba corresponde al editor, y nos permite colocar la cámara en cualquier momento donde deseemos haciendo *click* con el ratón. Esta vista es útil para tener un perspectiva de la escena completa, y poder observar como se desarrollan los algoritmos por ella. En la vista de abajo se encuentra la perspectiva desde el vehículo. Esta vista es útil cuando se realiza el seguimiento de la ruta, para poder observar el desplazamiento del vehículo.

Las dos columnas de la derecha corresponden a los elementos del proyecto. *Hierarchy* contiene los elementos de la escena que podemos seleccionar para modificarles, mientras que *Project* muestra todos los *assets* del proyecto que podemos usar para crear las distintas escenas.

Inspector mostrará los parámetros de los elementos seleccionados, y es donde vamos a modificar los distintos valores de los algoritmos. Seleccionando Coche en *Hierarchy* y nos mostrará los valores que podemos configurar en el proyecto:

[2]Repositorio del proyecto: https://github.com/vpe0001/Algoritmos_de_busqueda_3D-Unity

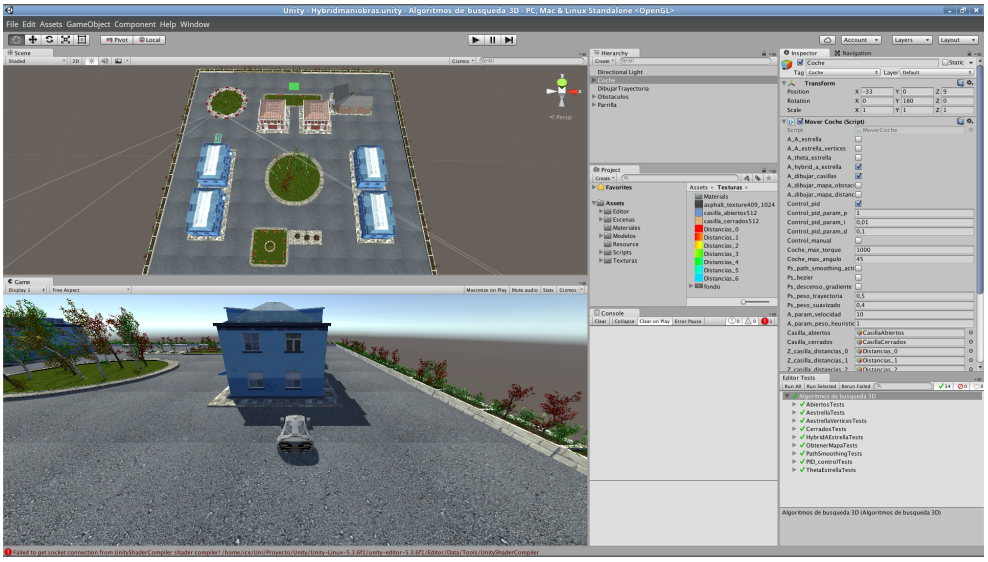

Figura E.1: Pantalla de la interfaz de *Unity*.

- ***Transform***: donde podemos modificar la posición (los valores de *Position*) y la orientación del vehículo (el valor y de *Rotation*). También está disponible para el resto de los elementos como los obstáculos

- A continuación aparecen los cuatro algoritmos disponibles para seleccionar: el A*, Theta*, A* con vértices y Hybrid A*.

- Las opciones de *dibujar mapa obstáculos* y *dibujar mapa distancias* muestran respectivos mapas al inicio de la ejecución. Aunque no es recomendable realizar la ejecución completa con los mapas activados puesto que, además de perder visibilidad, el rendimiento debido al gran número de elementos en pantalla se reduce considerablemente.

- La opción siguiente nos permite activar el *PID Controller* para el movimiento autónomo del vehículo, y podemos modificar los parámetros P, I y D para comprobar como afectan al movimiento.

- Control manual nos permite, una vez activado, manejar al vehículo con las teclas del teclado.

- Con *Path Smoothing* activamos el suavizado de la ruta. Si no activamos las opciones de *Descenso Gradiente* ni *Curvas Bézier*, realizará la eliminación del zigzag. Si no, realizará la opción seleccionada. También es posible modificar los valores de los parámetros del *Descenso gradiente* y ver las distintas rutas que genera.

- La opción de *param velocidad* se utiliza cuando no se ha seleccionado ni *PID control* ni *control manual*. En ese caso el vehículo no se moverá a usando las físicas, si no que se trasladará a través de la ruta hasta la meta a la velocidad que indica ese parámetro.
- El parámetro *peso heurísticas* sirve para configurar cuanta importancia tendrá la función H() en las rutas generadas en relación a la función G(). Si el valor es grande, el algoritmo búscará más en anchura, mientras que si es pequeño, intentará antes acercarse a la meta.

Manual para el ejecutable

La versión ejecutable es una aplicación reducida de lo que se puede hacer desde *Unity*. Consiste en un menú principal donde se pueden escoger ocho escenas preconfiguradas.

Para ejecutarlo se hace doble *click*, a continuación aparecen las opciones gráficas de resolución y preajustes de calidad de la imagen disponibles. Pulsando aceptar se inicia el menú de la aplicación, como se muestra en la figura E.2[3].

Figura E.2: Pantalla del menú del ejecutable.

Para usarlo, hacemos *click* en el botón correspondiente y se ejecutará la escena asociada. Cuando el vehículo llegue a la meta, la ejecución habrá terminado y se puede cerrar la ventana. Para cargar otra escena repetimos el proceso.

[3] *Unity* no permite el uso de acentos en los elementos de texto para la interfaz

La columna de botones de la izquierda corresponden al mismo escenario, pero cada uno con un algoritmo diferente para poder compararles. Los cuatro botones restantes corresponden al *Hybrid A**, donde el vehículo realiza distintas maniobras.

Bibliografía

[1] U. Technologies, "Unity 3d." [Online]. Available: https://unity3d.com/es/

[2] BlueRaja, "High speed priority queue for c#," 2016. [Online]. Available: https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp