UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Algoritmos de búsqueda 3D

Presentado por Víctor Pérez Esteban
en Universidad de Burgos — 6 de junio de 2017
Tutor: José Francisco Diez Pastor, César García Osorio

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. José Francisco Diez Pastor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Víctor Pérez Esteban, con DNI 71279579A, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 6 de junio de 2017

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En los últimos años hemos visto la evolución de autómatas que empiezan a hacerse presente en la sociedad. Uno de estos autómatas en lo que más se han observado esfuerzo en el desarrollo, inversiones por las grandes compañías y relevancia social ha sido la aparición de los primeros prototipos de los vehículos autónomos. Coches que se conducen solos.

Para la realización de estos autómatas se tienen que combinar muchos procedimientos que doten del conocimiento necesario al autómata sobre su entorno, y que permitan una vez adquirido su utilización para desplazarse a su destino. Dos de estos procedimientos principales son el cómo conocer la ruta a seguir, y una vez conocida la ruta cómo seguirla hasta alcanzar su final.

En este trabajo simularemos estos procesos en un entorno tridimensional, para poder simular algunos de los problemas que se encuentran los vehículos autónomos en el mundo real y tener una visión de como solventar los problemas que plantea. Se calcularán rutas que se puedan seguir por parte del vehículo y se implementarán métodos para seguir esas rutas de forma autónoma, viendo en cada paso los problemas que se plantean.

Descriptores

Búsqueda de rutas, vehículos autónomos, seguimiento de rutas ...

Abstract

In recent years we have seen the evolution of automatons that are beginning to be present in society. One of these automatons in what has seen the most effort in development, investments by large companies and social relevance has been the appearance of the first prototypes of autonomous vehicles. Cars that are driven by itselves.

For the realization of these automatons many procedures have to be combined to provide the automaton with the necessary knowledge about its environment and to allow it to be used once it has reached its destination. Two of these main procedures are how to know the route to follow, and once known the route how to follow it until reaching its end.

In this work, we will simulate these processes in a three-dimensional environment, in order to simulate some of the problems that autonomous vehicles find in the real world and to have a vision of how to solve the problems that it poses. Routes will be calculated that can be followed by the vehicle and will be implemented methods to follow these routes autonomously, seeing at each step the problems that arise.

Keywords

Routes searching, autonomous vehicles, tracking routes.

Índice general

Índice de figuras

Índice de tablas

Introducción

Descripción del contenido del trabajo y del estrucutra de la memoria y del resto de materiales entregados.

Objetivos del proyecto

Los objetivos del proyectos son:

1. Crear un espacio tridimensional dónde poder representar y probar el desarrollo de un vehículo autónomo.
2. Desarrollar algoritmos que permitan la búsqueda de rutas desde la posición del vehículo hasta una meta.
3. Dotar el vehículo de autonomía para poder seguir las rutas calculadas por si mismo.

En el proyecto se implementarán los algoritmos que hemos considerado más interesantes, y se plantea de tal forma que sea posible añadir en el futuro nuevos algoritmos para poder mejorar la autonomía del vehículo.

Conceptos teóricos

En aquellos proyectos que necesiten para su comprensión y desarrollo de unos conceptos teóricos de una determinada materia o de un determinado dominio de conocimiento, debe existir un apartado que sintetice dichos conceptos.

Algunos conceptos teóricos de LaTeX[1].

3.1. Algoritmo A*

A* es un algoritmo de búsqueda informada del tipo primero el mejor, que usa una función de evaluación para elegir hacia que nodo expandirse desde un nodo inicial hacia un nodo final.

Para representar el espacio de búsqueda del algoritmo, se usa una cuadrícula donde cada nodo puede representar un espacio donde es posible desplazarse o un obstáculo que es inalcanzable.

La función F() de evaluación calcula el coste de cada nodo, con lo que se eligen los nodos con menor coste para llegar al destino a través del camino óptimo. Esta función está formada por dos funciones a su vez. Una función G() que calcula el coste del camino seguido desde el nodo inicial a ese nodo concreto, y una función H() que hace un estimación del coste del camino desde ese nodo al nodo final o meta.

Al algoritmo comienza desde el nodo inicial explorando los nodos adyacentes o sucesores, cual es de menor coste. Un nodo ya explorado, es decir del que ya se ha buscado sus sucesores, se manda a la lista de nodos cerrados. Con los sucesores se forma una lista de nodos abiertos o por explorar, de la cual se elige el de menor coste para ser el siguiente en ser explorado, hasta

[1] Créditos a los proyectos de Álvaro López Cantero: Configurador de Presupuestos y Roberto Izquierdo Amo: PLQuiz

que se alcance el nodo meta o no queden más nodos por ser explorados. Si al explorar un nodo esta ya se encontraba en alguna de las listas de nodos abiertos o cerrados, se actualizarán los valores de los nodos al de menor coste encontrado.

Heurística

Al algoritmo A* es completo, lo que significa que encontrará un camino hasta la meta siempre que este exista. Además, para que sea admisible, que significa que siempre encontrará un camino óptimo, su función H() también debe ser admisible.

Una función H() es admisible siempre y cuando no sobreestime el coste del camino desde un nodo hasta la meta. Por ejemplo se puede considerar que el camino desde un nodo hasta la meta será la línea recta.

La admisibilidad del A* trae consigo un gran coste computacional debido al gran número de nodos explorados. Para mejorar la eficiencia podemos dar pesos a las funciones G() y H(), de tal forma que si damos mas valor a G() la búsqueda se expandirá en anchura buscando el camino, mientras que si damos más valor a H() se expandirá más rápido acercándose a la meta.

Pseudocódigo A*

Además de secciones tenemos subsecciones.

3.2. Pathsmoothing

Un problema del A* tradicional es que las rutas que encuentra, aunque sean las más cortas no tienen una apariencia realista. Esto es debido a que el A* discretiza el espacio de búsqueda, en nuestro caso el espacio en tres dimensiones, para poder buscar la ruta. Dependiendo de la representación que elijamos, se acercará más a o menos a la realidad, pero en cualquier caso se producirá una diferencia que suele estar alejada de la representación ideal de esa misma ruta.

Por ejemplo, en el caso de una búsqueda en un parrilla, si seguimos el camino a través de cada casilla, cuando el camino siga una ruta en diagonal, el A* seguirá un camino en zigzag. Aunque este camino es perfectamente válido, en la realidad no se sigue un camino en zigzag sino que se sigue la línea recta que representa. En el caso de las curvas el problema es parecido. El camino encontrado estará formado por segmentos rectos en vez de seguir una ruta redondeada.

Por este motivo una vez realiza una vez obtenida la ruta, un proceso de suavizado de tal forma que se acerque la ruta obtenida a través del A*, a la representación ideal de esa ruta.

Eliminar el zigzag

El primer método que hemos usado y que suele dar buenos resultados, es eliminar el zigzag. Este método es básicamente lo mismo que realiza el theta* que explicaremos más adelante.

El zigzag es el problema más habitual que nos hemos encontrado. Al representar un espacio de tres dimensiones a través de casillas, se produce habitualmente porque aunque se use la representación octal permitiendo el movimiento diagonal, esto al pasarlo a un espacio en tres dimensiones quiere decir que sólo tenemos ocho posibles ángulos de movimiento: 0^o, 45^o, 90^o, 135^o, 180^o, 225^o, 270^o y 315^o

En realidad las posibilidades reales para movernos son cualquier angulo de los 360^o. Por tanto, si nuestro objetivo está en un angulo diferente a esos ocho, se produce un zigzag combinandoles hasta que se consigue llegar.

La forma para eliminar el zigzag, es comprobar si es posible eliminar estos pasos intermedios. Es decir, si para llegar al objetivo el A* ha usado varias casillas en el espacio discreto, es posible que en un espacio no discreto pudiéramos llegar directamente con un movimiento en un ángulo distinto a los ocho que permite el A*.

Para ello, usamos la línea de visión. Si existe una línea de visión entre una casilla del A* y otra quiere decir que podemos desplazarnos siguiendo una línea recta formada entre esas dos casillas. Con las casillas devueltas por el A*, comprobamos si existe línea de visión entre ellas. Entre casillas consecutivas siempre habrá línea de visión, porque si no no sería un camino válido, pero puede también haber línea de visión entre las siguientes de forma que haya casillas sobrantes debido al zigzag y a la forma que tiene el A* de buscar el camino.

Así que lo que comprobamos es que existe una línea de visión entre una casilla del A* y la siguiente más lejana posible, y eliminamos a las casillas intermedias. De esta forma obtenemos una línea recta entre esas dos casillas y eliminamos el zigzag producido por las casillas intermedias.

Descenso gradiente

El descenso gradiente es un algoritmo iterativo que a partir de unos valores iniciales, itera una función modificando esos valores hacía un mínimo de esa función. El algoritmo termina cuando el cambio que se producen en los valores es menor que la tolerancia indicada.

Para optimizar la ruta, queremos obtener los valores que minimizan las distancia entre los puntos sucesivos de la ruta, y a su vez que minimicen las distancia a la ruta original.

Para minimizar la distancia entre los puntos con respecto a la ruta original usaremos las función:

$$y_i = y_i + \alpha(x_i - y_i)$$

Donde α es el peso que le damos a cuanto de cerca queremos que la ruta suavizada esté de la ruta original, x es el punto de la ruta original y y es el nuevo punto suavizado.

Para minimizar la distancia entre los puntos sucesivos de la ruta, usamos la función:

$$y_i = y_i + \beta(y_{i+1} + y_{i-1} - 2 * y_i)$$

Donde β es el peso que le damos al suavizado de la ruta, y tenemos en cuenta la distancia tanto con el punto anterior como con el puto siguiente.

Para calcular el error que se produce, o el cambio hacia el mínimo, usamos la función:

$$error = error + (z_i - y_i)$$

Donde z es el valor de y antes de calcular el nuevo mínimo.

Curvas Bézier

Las curvas Bézier [4] [6] son una forma de representar curvas a través de una función que recibe un parámetro. Hemos usado la función de las curvas de Bézier para suavizar la ruta y así obtener giros y curvas más cercanas a una representación real

Una función de una curva Bézier tiene varios, al menos dos, puntos de control. De estos puntos, el primero y el último representan el inicio y el final de la curva. Esta función además recibe un parámetro que puede tomar los valores comprendidos entre cero y uno. Si el parámetro toma el valor cero, entonces la función devuelve el punto de inicio, mientras que si toma el valor uno devuelve el punto final. De esta forma, dando valores entre cero y uno, la función devuelve los valores que se encuentran entre los puntos de inicio y de fin.

Si la función toma dos puntos de referencia, lo que obtendremos sera una recta y sus puntos intermedios. Si la función toma tres puntos entonces tendremos una curva donde el vértice será cercano al punto intermedio. Podemos

usar más puntos para representar curvas más complejas o dar curvas con más variedad de formas.

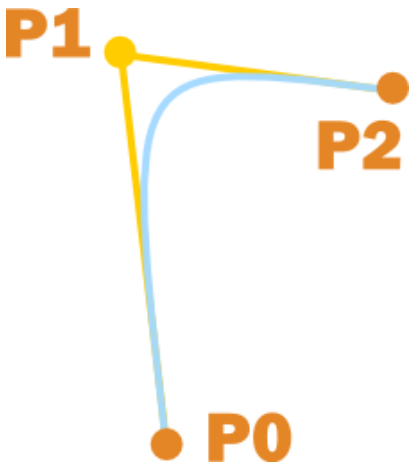

Figura 3.1: Esquema de una curva Bézier cuadrática de Herman Tulleken, Bézier Curves for your Games: A Tutorial [5].

Hemos usado la función Bézier con tres puntos después de eliminar el zigzag, así que cada tres puntos del A* sin zigzag se usan como puntos de control que servirán para crear la curva. Si los puntos más o menos están alineados, el resultado será una recta suavizada, y si forman una curva, se eliminan los segmentos rectos y se reemplazan con puntos que forman una curva.

La función Bézier cuadrática para tres puntos de control y parámetro t es:

$$[x, y, z] = (1–t)^2 P_0 + 2(1–t)tP_1 + t^2 P_2$$

En formato expandido:

$$x = (1–t)^2 x_0 + 2(1–t)tx_1 + t^2 x_2$$
$$y = (1–t)^2 y_0 + 2(1–t)ty_1 + t^2 y_2$$
$$z = (1–t)^2 z_0 + 2(1–t)tz_1 + t^2 z_2$$

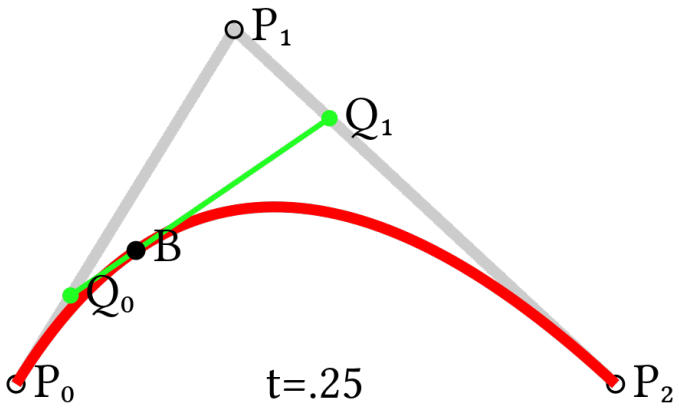

Figura 3.2: Esquema de una curva Bézier cuadrática de Bézier curve — Wikipedia [7].

3.3. Theta*

Theta A* es un algoritmo desarrollado por A. Nash, K. Daniel, S. Koenig y A. Felner [3] [1] que modifica el A* original para obtener rutas más realistas.

Como explicamos anteriormente, el A* por la forma en la que representa el espacio de búsqueda da resultados poco realistas, con lo que hay que realizar un postprocesado de la ruta obtenida. El theta* incorpora este proceso al propio A* para obtener de forma directa rutas que no requieran del proceso de suavizado o que lo minimice.

Para ello, el cambio que hace el Theta* respecto al A* es que un sucesor a una casilla puede ser cualquier casilla, en vez de ser unicamente las que tiene alrededor. El algoritmo es el mismo que el A* original, pero cuando se calculan loa sucesores, además de calcular si es un sucesor válido y el coste de ese sucesor, comprueba si hay linea de visión con la casilla del nodo padre

del nodo que lo generó. Si no hay línea de visión, sigue el algoritmo del A* y le asigna al sucesor como padre la casilla que lo originó. Pero si hay línea de visión, le asigna como padre no el nodo que lo generó, sino el padre de este, calculando el coste del sucesor teniendo en cuenta esta otra casilla. De esta forma elimina las casillas intermedias que origina el A* que no son necesarias de forma similar a como se realizaría en el post procesado para eliminar el zigzag.

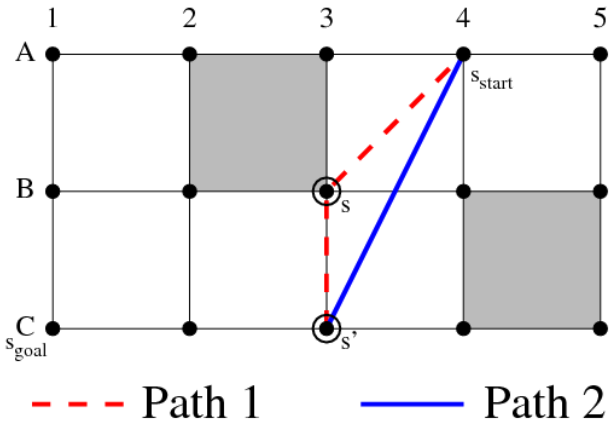

Figura 3.3: Comparacion de los caminos elegidos por el A* (path1) y el theta* (path2) de [2].

Pseudocódigo Theta*

3.4. A* con vértices

Comparativa

3.5. Imágenes

Se pueden incluir imágenes con los comandos standard de LaTeX, pero esta plantilla dispone de comandos propios como por ejemplo el siguiente:

Figura 3.4: Autómata para una expresión vacía

3.6. Listas de items

Existen tres posibilidades:

- primer item.
- segundo item.

1. primer item.
2. segundo item.

Herramientas	App AngularJS	API REST	BD	Memoria
HTML5	X			
CSS3	X			
BOOTSTRAP	X			
JavaScript	X			
AngularJS	X			
Bower	X			
PHP		X		
Karma + Jasmine	X			
Slim framework		X		
Idiorm		X		
Composer		X		
JSON	X	X		
PhpStorm	X	X		
MySQL			X	
PhpMyAdmin			X	
Git + BitBucket	X	X	X	X
MikTEX				X
TEXMaker				X
Astah				X
Balsamiq Mockups	X			
VersionOne	X	X	X	X

Tabla 3.1: Herramientas y tecnologías utilizadas en cada parte del proyecto

Primer item más información sobre el primer item.

Segundo item más información sobre el segundo item.

-

3.7. Tablas

Igualmente se pueden usar los comandos específicos de LATEXo bien usar alguno de los comandos de la plantilla.

Técnicas y herramientas

Esta parte de la memoria tiene como objetivo presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto. Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso somero a los fundamentos esenciales y referencias bibliográficas para que el lector pueda ampliar su conocimiento sobre el tema.

4.1. Programas usados

Unity

Unity un motor para juegos usado principalmente para desarrollar viejuegos y simulaciones. Que sea un motor para juegos quiere decior que está formado por varios motores a su vez, como el motor gráfico y el motor de físicas.

Unity proporciona un editor donde se puede crear escenas junto con un gran número de herramientas, ejemplos y modelos para crear escenarios tanto 2D y principalment 3D.

Tiene una licencia de uso por pasos de ingresos obtenidos por el contenido creado con el. En la versión Personal, su puede usar libremente siempre que los ingresos generados por el contenido usado no supere los 100 dólares anuales, y es la versión de la licencia usada.

Para el proyecto probamos de forma descente las distintas versiones de Unity para linux en un xUbuntu 16.04, debido a que las últimas versiones producían errores que cerraban el editor al iniciarlo o al poco tiempo despues

de abrirse. La primera versión estable que funcionaba en el equipo de desarrollo fue la versión 5.3.6f1, y ha sido la usada para su realización.

Texmaker

Para la realización de la memoria se ha usado latex con el editor texmaker. Texmaker es un editor libre con licencia GPL.

Texmaker incluye varias herramientas como el corrector ortográfico o el visor de pdf que facilitan la creación de los documentos.

La versión usada ha sido la 4.4.1.

SmartGit

SmartGit es un gestor gráfico para git que soporta github y que facilita la realización de commits, push y pull de los repositorios del proyecto.

Tiene una licencia no comercial que permite el uso de forma gratuita a desarrolladores de código abierto, profesores, estudiantes, desarrollos no comerciales y también organizaciones sin ánimo de lucro.

La versión que hemos usado ha sido la 17.0.3

GitHub y ZenHub

La plataforma elegida para los repositorios ha sido github, principalmente por su compatibilidad con Zenhub que es la herramienta de gestión de proyectos que hemos usado.

Zenhub es una herramienta que permite la planificación y control del proyecto. Es un complemento de firefox que se integra con github y añade funciones como las boards para manejar el control de las issues y las milestones del proyecto, y que permite ver los gráficos del progreso y burnouts que se ha realizado.

La versión utilizada ha sido la 2.34.2

Aspectos relevantes del desarrollo del proyecto

5.1. Nav Mesh

Una nav mesh es un conjunto de polígonos, triángulos normalmente, que se usan en los motores gráficos 3D para representar la zona o camino recorrible de un agente. Se crean a partir de los objetos estáticos de una escena, que son los obstaculos, donde no se puede desplazarse. El resto de la zona de la escena será zona recorrible donde nos podemos desplazar.

En relación por ejemplo al A*, sería el equivalente a la matriz que reprsenta el mapa donde se bucará los nodos a explorar.

Está formada por los vértices de los polígonos, y si un punto esta dentro del área que forman entonces ese punto es recorrible. Esto es mas realista en un escenario en tres dimensiones debido a que permite un movimiento menos discretizado, al contrario que las parrillas de casillas que se adecuan más a una representación en dos dimensiones. Y si consideramas los vértices los nodos sucesores, resulta más óptimo computacionalmente, ya que el número de nodos (vértices) a explorar es menor, debido a que no tenemos que tener en cuenta todos los puntos intermedios hasta llegar al vértice, si no unicamente si el vértice es visible o alcanzable desde el nodo actual.

Hemos usado una nav mesh para generar de forma automática el mapa para el A*. En Unity esto se puede hacer de dos formas. En versiones anteriores a la versión 5.6 solo es posible generarlo desde el editor antes del tiempo de ejecución. A partir de la versión 5.6, que permite acceder a las herramientas del editor en tiempo de ejecución, se puede construir a través de NavMeshBuilder, dando mucha más libertad a la hora de crear mapas de forma dinámica o interactiva. Lamentablemnte no se pudo usar la versión 5.6 y la versión mñas actual que funcionba en el equipo de desarrollo fue la 5.3.6.

La clase NavMesh de Unity tiene métodos que podemos usar como RayTrace, que lanza un rayo desde un vector a otro y devuelve si son visibles dentro de la NavMesh, es decir, si existe una línea recta entre ellos que los una dentro del camino recorrible. Este métodol lo hemos usado para decidir si un nodo sucesor es válido o no. Al usarlo encontramos un bug que producía que a veces no se encontrará el camino o bucles infinitos a recorrer las listas. Esto es debido a que puede ocurrir que no devuelva lo mismo dependiendo del orden o sentido en el que se le indiquen los vectores entre los que lanzar el rayo. Por ejemplo RayTrace(vector1, vector2) puede ser visible pero RayTrace(vector2, vector1) no. Para solucionarlo decidimos que para que fuera un sucesor valido debie ser visible en los dos sentidos.

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía

[1] Alex Nash. Theta*: Any-angle path planning for smoother trajectories in continuous environments.

[2] Alex Nash. Theta*: Any-angle path planning for smoother trajectories in continuous environments. figure 6 comparison theta* and a*.

[3] Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids.

[4] Herman Tulleken. Bézier curves for your games: A tutorial.

[5] Herman Tulleken. Bézier curves for your games: A tutorial. quadratic curve.

[6] Wikipedia. Bézier curve — wikipedia, the free encyclopedia, 2017.

[7] Wikipedia. Construction of a quadratic bézier curve, 2017.