Grado en Matemática Computacional

Proyecto Final de Grado

Modelización Matemática de objetos (sprites) en videojuegos

Autor:
Jorge González López

Tutor académico:
Vicente Palmer Andreu

Fecha de lectura: 20 de Noviembre de 2014
Curso académico 2013/2014

Resumen

Durante mi estancia en prácticas, mi principal objetivo ha sido desarrollar un videojuego con diferentes minijuegos incorporados , que sirva para promocionar a la ciudad de Barcelona como reclamo turístico. La principal funcionalidad del juego era programar los diferentes algoritmos informáticos incluyendo el diseño de sus objetos y sus sprites asociados. Por ello este trabajo de final de grado tiene como objetivo construir la imagen de un boomerang, como ejemplo de diseño en un lenguaje de bajo nivel de uno de estos objetos y sus correspondientes posiciones en las escenas del videojuego llamadas sprites. Este diseño se basa en el uso de diferentes herramientas matemáticas. La primera de estas herramientas es la interpolación lineal usando polinomios de Lagrange y la segunda herramienta será a través de la aproximación utilizando polinomios de Bernstein. De este modo, podremos reproducir el movimiento del boomerang y asi poder visualizarlo utilizando una herramienta de diagramas de flujo que anteriormente he utilizado en mi estancia en prácticas.

Cabe destacar que la elección de la construcción del sprite del boomerang fue debida a que se quería obtener un proyecto en el que existiera un nexo de unión entre la estancia en prácticas y el ámbito matemático, que en este caso se trataba de la rama de Geometría. Tras muchos debates se llegó a la conclusión de que se podía utilizar el diseño geométrico para construir un objeto interpolado que posteriormente se podría utilizar en la herramienta informática. Como se estudió las diferentes maneras de interpolar objetos a través de polinomios de Lagrange y Bernstein se concluyó que un boomerang era un objeto idóneo para reflejar este nexo de unión.

Palabras clave

Sprite, Multiplataforma, Caja de Colisiones, Polinomios de Lagrange, Polinomios de Bernstein, Curvas de Bézier.

Keywords

Sprite, Multiplatform, Collision Box, Lagrange Polynomials, Bernstein Polynomials, Bézier Curves.

Índice general

1. Introducción **5**

1.1. Contexto y motivación del proyecto 5

2. Descripción del proyecto de la Estancia en Prácticas **7**

2.1. Objetivos del proyecto 7

2.2. Metodología y definición de tareas 7

2.3. Planificación temporal de las tareas 8

2.4. Trabajo Realizado en La Empresa 8

2.4.1. Una Herramienta de diseño de videojuegos: Gamesonomy 8

2.4.2. La Escena 9

2.4.3. Los Actores 9

2.4.4. Las Reglas 12

2.4.5. Explicación Funcionalidad del Juego 21

3. Modelización Matemática de los Objetos **27**

3.1. Introducción y objetivos 27

3.2. Interpolación con polinomios de Lagrange 28

3.2.1. Definición de los Polinomios de Lagrange 28

3.2.2. Propiedades de los Polinomios de Lagrange 29

3.2.3. Construcción del sprite mediante Polinomios de Lagrange 30

3.3. Interpolación con Curvas de Bézier 34

3.3.1. Introducción 34

3.3.2. Algoritmo de Casteljau 34

3.3.3. Polinomios de Bernstein 38

3.3.4. Propiedades de las Curvas de Bezier 46

3.3.5. Construcción del sprite mediante Curvas de Bézier 48

3.4. Visualización del Sprite 58

4. Conclusiones **59**

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Mi estancia en prácticas tuvo lugar en el Instituto de Nuevas Imágenes y Tecnología (INIT) y el principal objetivo en esta, era crear un videojuego en multiplataforma (tanto en Android como en iOS), de manera que este videjuego estuviese ambientado en las distintas localizaciones de la ciudad de Barcelona.

Una vez acabada mi estancia en prácticas, mi tutor y yo pensamos sobre que temática relacionada con las matemáticas, se podria desarrollar en el Trabajo Final de Grado. Esta temática deberia estar relacionada con lo aprendido en la empresa a través de la herramienta de Gamesonomy, que es una herramienta básica de diseño y creación de juegos.

En el videojuego de Gamesonomy se dota de movimiento a distintos objetos y a sus distintas posiciones (llamadas sprites). El objeto de trabajo es la descripción matemática del diseño de estos objetos usando distintos métodos: los polinomios de Lagrange y los polinomios de Bernstein, que modelizan el contorno del sprite.

Tras un intenso estudio de todas la posibilidades, decidimos crear un videojuego que tuviera como actor principal un boomerang. El motivo de elegir un boomerang, fue que queríamos obtener un nexo de unión entre lo desarrollado en la estancia en prácticas y lo investigado en lo referente al ámbito matemático, y se vio que la interpolación de boomerang (por su sencillez) podía reflejar ambos aspectos de una manera idónea .

Respecto al ámbito informático del videojuego, el principal objetivo de este sera lanzar el boomerang a lo largo del escenario e intentar recoger todas las monedas posibles en el menor tiempo posible.

Respecto al ámbito matemático del videojuego, hay que destacar varios cosas:

1. Se usarán dos métodos para la construcción del sprite del boomerang. La primera construcción se realizará mediante interpolación utilizando polinomios de Lagrange y la segunda construcción se realizara mediante aproximación, utilizando polinomios de Bernstein.

2. Estas dos modelizaciones se realizaran 4 veces, con distintos puntos, para así conseguir o lograr diferentes sprites y poder realizar un animación adecuada del boomerang rotando sobre su eje central.

3. Una vez realizada las dos interpolaciones se estudiara el coste computacional de ambas y se estudiaran los resultados obtenidos.

Capítulo 2

Descripción del proyecto de la Estancia en Prácticas

2.1. Objetivos del proyecto

El Instituto de Nuevas Imágenes y Tecnología (INIT) fue el lugar donde realize mi estancia en prácticas de tal manera que pudiera ganar experiencia sobre el funcionamiento de una empresa, y así facilitar mi posterior inserción laboral. Cabe destacar que elegí esta empresa debido a que estaba ligada a la rama de Geometría.

El principal objetivo de esta estancia en prácticas, era realizar un videojuego en multiplataforma(Android e IOS). Dicho videojuego estará basado en la temática de Barcelona y servirá como reclamo turístico en dicha ciudad, promoviendo y enseñando de una manera didáctica, las distintas posibilidades turísticas que ofrece.

2.2. Metodología y definición de tareas

Este videojuego contendrá seis puntos claves como ruta turística de Barcelona. En cada punto el jugador dispondrá de un minijuego basado en la temática de dicho punto. Una vez el jugador haya conseguido pasarse dicho minijuego, obtendrá una serie de información característica de ese punto para que así fomente su visita turística.

Estos 6 minijuegos estarán basados en los siguientes puntos: las Ramblas, Parque Güell, Montjuïc, Camp Nou, Sagrada Familia y Casa Batlló, y tendrán como objetivo enseñar didácticamente las características turísticas de dichos sitios.

Cabe destacar que dichos minijuegos deben estar relacionados con su propio puntos turísticos y además deben facilitar la comprensión de información turística y el aprendizaje sobre estos.

Para la programación de dicho videojuegos se utilizará una herramienta propia de Gamesonomy, la cual esta basada en una serie de diagramas de flujo en los que intervienen una serie de actores y condiciones a los que se le aplica.

Una vez comprendido todo el funcionamiento y todas las posibilidades que ofrece esta herramienta, se debe realizar dichos minijuegos, de manera que estos, resulten didácticamente sencillos.

Además se debe procurar que la programación de dichos minijuegos sea los más eficiente posible para así evitar un sobrecoste en ellos, el cual podría ser responsable de un almacenamiento mayor de lo esperado.

2.3. Planificación temporal de las tareas

Fecha inicial	Fecha final	Tarea
25 febrero	9 marzo	Compresión y utilización adecuada de la herramienta
10 marzo	30 marzo	Realización de los 2 primeros minijuegos
31 marzo	20 abril	Realización de los 2 siguientes minijuegos
21 abril	18 mayo	Realización de los 2 últimos minijuegos
19 mayo	25 mayo	Montar el menú correspondiente al juego entero

Cuadro 2.1: Planificación Temporal de Las Tareas

2.4. Trabajo Realizado en La Empresa

Para poder entender mejor lo que he ido realizando en mi estancia en prácticas a lo largo de 3 meses, voy a hacer una introducción con los elementos clave e interfaces del programa u herramienta con el que he trabajado (Gamesonomy).

2.4.1. Una Herramienta de diseño de videojuegos: Gamesonomy

Gamesonomy es una herramienta básica de diseño y creación de juegos. Un usuario sin conocimientos de programación podrá diseñar fácilmente juegos formados por **escenas** creadas a partir de **actores** y **reglas** que condicionan su comportamiento.

2.4.2. La Escena

Inicialmente el juego aparece con una única escena vacía que se tendra que ir configurando.

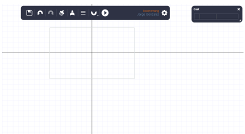

Figura 2.1: Escena

2.4.3. Los Actores

Los actores son los elementos u objetos que forman las escenas. Dentro de estas podemos encontrar los sprites, que son las diferentes imágenes asociadas a un actor y a sus movimientos.

2.4.3.1. Propiedades de un Actor

En este apartado vamos a mostrar las diferentes propiedades que posee un actor, tal y como aparecen en el interfaz de usuario correspondiente en el programa Gamesonomy y que mostraremos en la Fig 2.2, 2.3 y 2.4. Los principales atributos que definen a un actor son:

1. Información básica: establece la posición, tamaño, factor de escalado y rotación del actor. Además, permite establecer la forma de la caja de colisión e indicar si el actor está fijo en la escena. En la imagen descrita a continuación mostramos todos los elementos que pertenecen a la pestaña de información básica dentro del interfaz asociado al actor: posición, tamaño, factor de escalado, rotación del actor y pantalla.

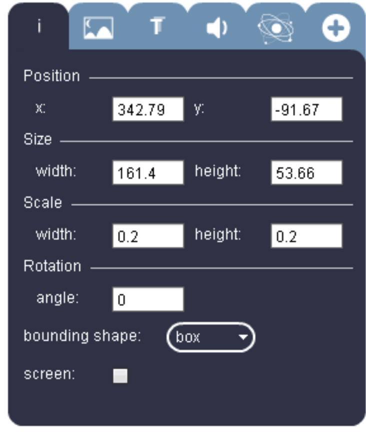

Figura 2.2: Información Básica

Los elementos siguientes definen la imagen descrita anteriormente:

- Position x/y: posición del actor en x e y.
- Size width/height: dimensiones ancho/alto originales del actor.
- Scale width/height: factor de escalado del actor en los ejes x e y respectivamente.
- Rotation angle: ángulo de rotación respecto del original, que se aplica al actor al representarse en la escena.
- Bounding shape: permite elegir entre una caja de colisión rectangular o esférica.
- Screen: si esta casilla está activa el actor estará siempre visible sin que le afecte un cambio en la posición de la cámara. Si la casilla no está activa, la posición del actor cambia si cambia la posición de la cámara.

2.4.3.1.2. Información sobre la visualización del actor: establece si el actor está visible en la escena, la imagen que lo representa, su color, su opacidad, y si el actor está invertido respecto a alguno de los ejes. En la imagen descrita a continuación, correspodiente al interfaz del actor (Fig 2.3), mostramos los elementos que permiten visualizar al actor.

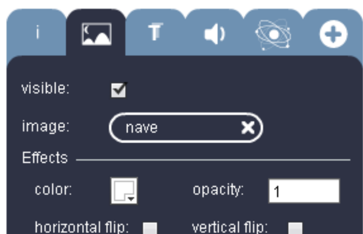

Figura 2.3: Visualización del Actor

2.4.3.1.3. Propiedades físicas del actor: habilita/deshabilita la aplicación de las propiedades físicas del actor, estableciendo si el actor es móvil, su peso, la fricción que ejerce, su rebote, su rozamiento en la escena y su movimiento. En la imagen descrita a continuación, correspodiente al interfaz del actor (Fig 2.4), mostramos todos los elementos que pertenecen a la pestaña propiedades físicas que posee un actor.

Figura 2.4: Propiedades Físicas

Los elementos siguientes definen la imagen descrita anteriormente:

1. Active: habilita/deshabilita las propiedades físicas del actor.
2. Type: especifica si el actor se mueve o permanece fijo.
   - Static: el actor permanece inmóvil aunque se le aplique una velocidad o el resultado de una fuerza (colisión). Esta opción es necesaria tenerla activa para evitar que cuando otro actor choque contra él, éste se mueva.
   - Kinematic: el actor es móvil únicamente cuando se le aplica una velocidad.
   - Dynamic: el actor es móvil siempre, tanto si se le aplica una velocidad o una fuerza.
3. Movement fixed rotation: establece si el actor gira. Si la casilla está activa, significa que el actor no gira.
4. Velocity X / Velocity Y: establece la velocidad lineal en xo en y del actor.
5. Angular: establece la velocidad angular del actor.
6. Material density: establece el peso del actor. A mayor valor, mayor peso.
7. Friction: establece si se va a tener en cuenta la fricción de un actor con algún otro en la escena.
8. Restitution: establece si el actor tiene rebote. A mayor valor, mayor rebote. Al ser un objeto físico, se comportará como un objeto real, por lo que el número de rebotes no se puede establecer en un número determinado.
9. Damping linear/angular: establece el rozamiento lineal/angular que ejerce el actor en la escena cuando no hay nada.

2.4.4. Las Reglas

Un actor puede tener asociado un comportamiento definido a partir de una serie de reglas establecidas en el editor de reglas. Las reglas regulan las acciones que el actor ejecuta. Estas reglas las vamos a mostrar en las figuras siguientes tal y como aparecen en el interfaz correspondiente al Editor de Reglas en el programa (ver Fig 2.5).

2.4.4.1. El Editor de Reglas

En el editor de reglas que mostramos a continuación (Fig 2.5), se muestran todos los diferentes tipos de reglas que este tipo de herramienta posee.

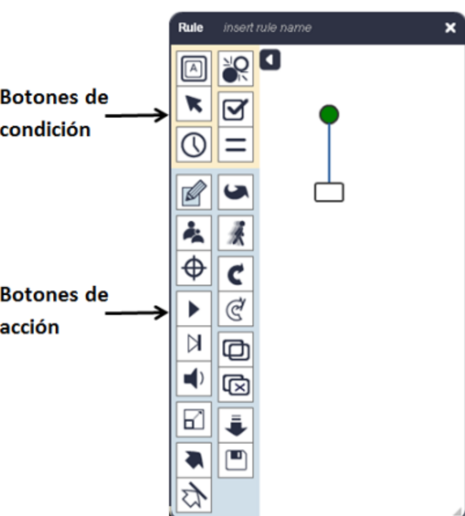

Figura 2.5: Editor de Reglas

Las reglas, están representadas gráficamente mediante un diagrama de flujo o grafo creado a partir de la selección de condiciones y acciones.

Cabe destacar que sobre este diagrama de flujo podemos añadir las diferentes acciones o condiciones de la herramienta.

Para colocar una condición o una acción en el grafo, basta con arrastrar el botón correspondiente desde la barra de botones lateral, al lugar del grafo donde se quiera colocar.

Al insertar una condición, se crean automáticamente dos salidas. La salida derecha se ejecutará si una vez evaluada la condición, el resultado es verdadero. Si el resultado de evaluar la condición fuese falso, se ejecutará la salida izquierda.

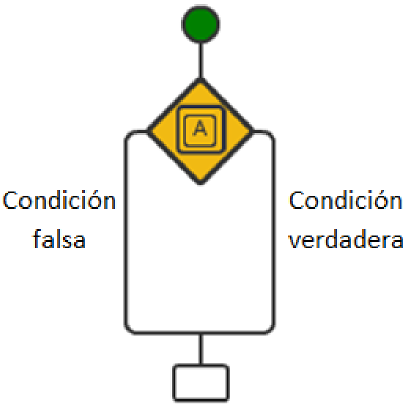

Figura 2.6: Grafo sobre las Reglas

2.4.4.2. Botones de Condición

Una vez colocada una condición en el editor de reglas, haciendo clic sobre el icono incrustado en el grafo, aparecen las opciones que permiten configurar esta condición. En este apartado vamos a mostrar todas las condiciones que esta herramienta nos proporciona y su imagen asociada o explicativa.

Key: evalúa el estado de una tecla del teclado. La casilla KeyMode ofrece la posibilidad de elegir entre las opciones keydown (tecla pulsada), keyup (tecla no pulsada) y Keypress (tecla pulsada y mantenida).

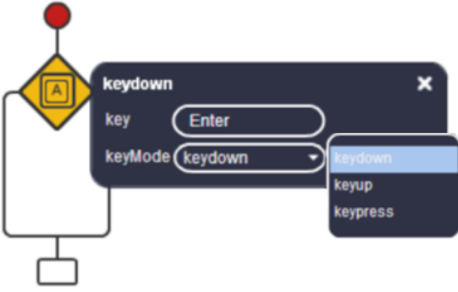

Figura 2.7: Regla Key

Mouse: evalúa el estado del botón izquierdo ratón. Para configurar esta condición, la casilla mode ofrece la posibilidad de elegir entre mousedown (botón del ratón pulsado), mouseup (botón del ratón no pulsado), isovers (ratón pasando por encima del objeto al que se le aplica la regla) y click (se hace click sobre él).

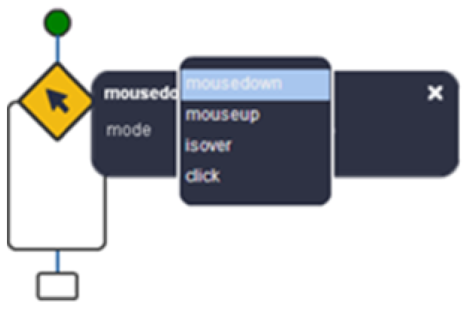

Figura 2.8: Regla Mouse

Timer: establece un periodo de tiempo determinado. Su funcionamiento consiste en realizar la acción establecida en la salida derecha de la condición cada espacio de tiempo especificado en la casilla seconds y la acción establecida en la salida izquierda de la condición el resto del tiempo.

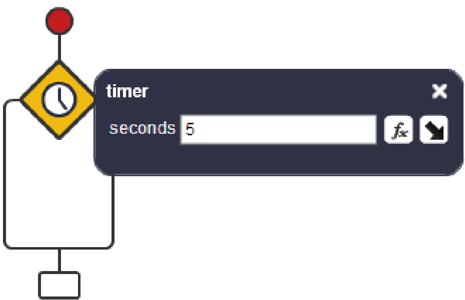

Figura 2.9: Regla Timer

Collides: controla si el actor colisiona con algún actor o actores de la escena, seleccionando la etiqueta del actor con el que se controlará su colisión en la casilla tag. Hay que ser cuidadoso a la hora de asignar etiquetas, ya que a un mismo actor se le pueden asignar más de una etiqueta y varios actores pueden tener asignada una única etiqueta.

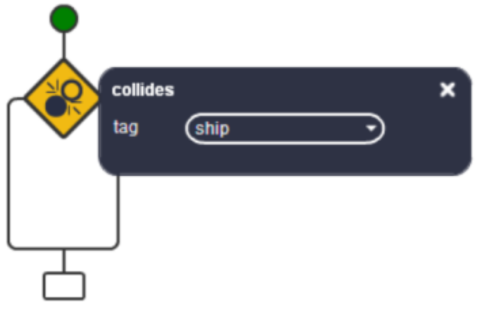

Figura 2.10: Regla Collides

Check: evalúa una propiedad booleana (verdadero/falso) de un actor, del juego o de la escena, incluida en la casilla property.

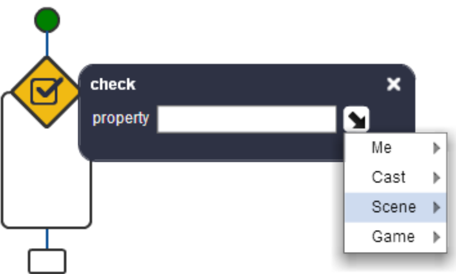

Figura 2.11: Regla Check

Compare: compara los dos operandos introducidos en las casillas en blanco.

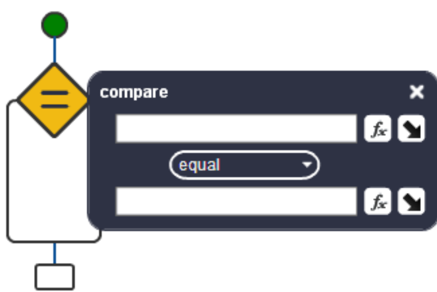

Figura 2.12: Regla Compare

Las diferentes opciones de comparación que pueden realizarse entre los operandos incluidos en estas casillas son: equal, different, less, greater, less-equal, greater-equal.

2.4.4.3. Botones de Acción

Las acciones pueden ejecutarse después de evaluar si una condición se cumple o no, o se realizan siempre sin necesidad de que se establezca condición alguna. En este apartado vamos a mostrar todas las acciones que esta herramienta nos proporciona y su imagen asociada o explicativa.

Change: cambia el valor de una propiedad de un actor introducida en la casilla property por el valor introducido en la casilla value.

Figura 2.13: Acción Change

Spawn: muestra en la escena un actor que está creado con anterioridad.

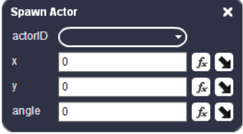

Figura 2.14: Acción Spawn

Destroy: destruye el actor, lo elimina de la ventana play. Cuando se elimina un actor del juego, hay que tener en cuenta si está asociado a otro actor en el juego, por ejemplo, si se quiere realizar un spawn de él posteriormente, no se podrá.

Translate: traslada el actor los píxeles indicados en la casilla step y el ángulo indicado en la casilla angle.

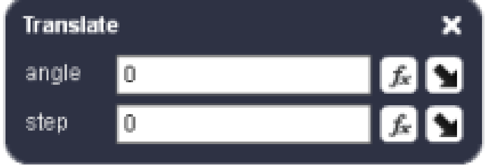

Figura 2.15: Acción Translate

Translate to: traslada el actor hacia el punto de coordenadas (x,y) indicadas en las casillas posX y posY, el desplazamiento en píxeles indicado en la casilla step.

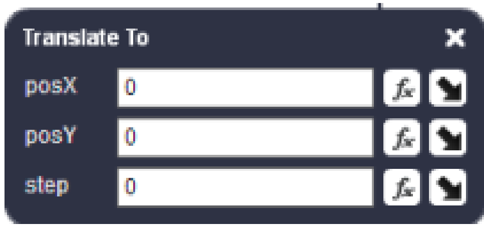

Figura 2.16: Acción Translate to

Scale: escala un actor respecto a las coordenadas (x, y) indicadas en pivotX y en pivotY. En scaleX / scaleY se introduce el factor de escalado en x y en y.

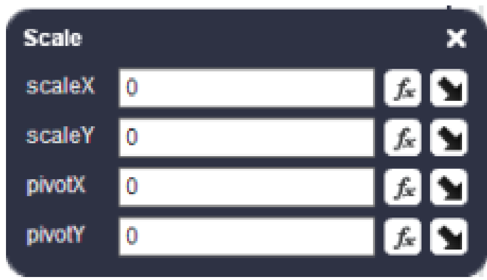

Figura 2.17: Acción Scale

Apply force: aplica una fuerza lineal indicada en la casilla strength con un ángulo indicado en la casilla angle.

Figura 2.18: Acción Apply Force

Apply force to: aplica la fuerza lineal especificada en la casilla strength, desde el actor, al punto indicado en las coordenadas (x, y) introducidas en las casillas x e y respectivamente. Con estos valores y la posición del actor se calcula el ángulo con el que se aplica la fuerza.

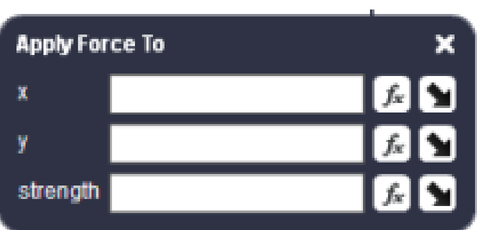

Figura 2.19: Acción Apply force to

Apply torque: aplica una fuerza con el ángulo indicado en la casilla angle.

Figura 2.20: Acción Apply torque

Animate: Crea la animación de un actor. Cada imagen que forma la animación, se incluirá con el botón . Una vez incluidas las imágenes, en la casilla fps se indicará cuántos frames por segundo se quieren mostrar.

Figura 2.21: Acción Animate

Rotate: permite rotar un actor el ángulo introducido en la casilla angle. La rotación se hace respecto a las coordenadas (x, y) indicadas en pivotX y pivotY respectivamente.

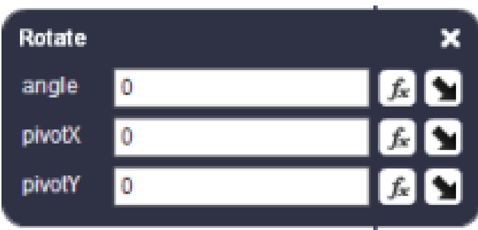

Figura 2.22: Acción Rotate

Rotate to: rota un actor hacia una posición de coordenadas (x, y) de la escena indicada en las casillas x e y, respecto al punto de coordenadas (x, y) indicado en pivotX y pivotY. En step se incluirá el número de intervalos en los que se va a recorrer el ángulo que se quiere rotar.

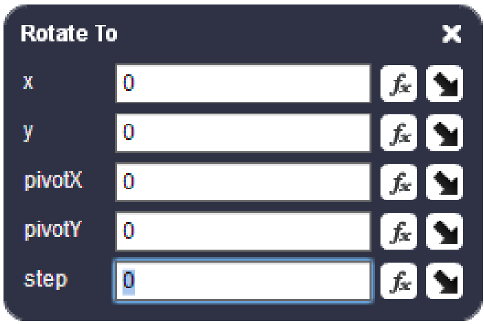

Figura 2.23: Acción Rotate to

New layer: sobre la escena activa se carga la escena (capa) especificada en scene.

Figura 2.24: Acción New Layer

Destroy layer: elimina la capa superpuesta para que aparezca la escena en la que estábamos situados antes del parar la ejecución del juego, dejando el juego situado en el momento en que se realizó la interrupción.

save: guarda en disco el valor de una propiedad de un actor, de una escena o del juego introducida en la casilla property con el nombre indicado en la casilla name.

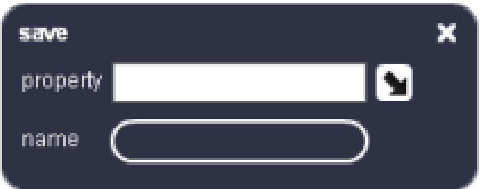

Figura 2.25: Acción Save

load: carga en el juego el valor de una variable salvada previamente en disco por medio de la acción save.

Figura 2.26: Acción Load

Una vez hemos introducido todas las funcionalidades que presenta esta herramienta, voy a comentar y presentar los diferentes juegos que he ido implementando en la empresa y los diferentes algoritmos y comportamientos en algunos de los actores de cada uno de los juegos. Hay que recordar que mi principal función en la empresa fue desarrollar un videojuego (dividido en 6 minijuegos) que sirviera como reclamo turístico de la ciudad de Barcelona.

2.4.5. Explicación Funcionalidad del Juego

El juego esta dividido en diferentes escenas. La primera es una pantalla de inicio en la que se introduce el juego. Una vez pulsado en el botón de play de esta pantalla, el juego nos llevara a otra escena que esta formada por una mapa turístico de Barcelona. En esta escena hay 6 puntos que representan cada uno de los juegos relacionados con su ámbito turístico.

Cabe destacar que en un primer momento no se disponen de todos los juegos, ya que el objetivo es que se vayan desbloqueando conforme se vayan logrando los objetivos de los anteriores minijuegos. Una vez elegido un minijuego, nos aparecera una pantalla de carga donde se nos explicara diferente información turística sobre el punto elegido. Una vez completada la carga del juego, comenzaremos a probar el juego, el cual nos dara un objetivo a lograr para desbloquear el siguiente minijuego. Si lo conseguimos lograr volveremos a la escena del mapa de Barcelona y podremos continuar con el posterior minijuego.

A continuación explicaremos la funcionalidad de cada uno de estos juegos más concretamente y los algoritmos utilizados para su eficiente creación.

Para no sobrecargar las imagenes a lo largo del proyecto cogeremos uno de los 6 minijuegos como ejemplo y mostraremos los gráficos asociados a dichos algoritmos que actuan sobre los sprites de dichos actores. En este caso el minijuego que vamos a utilzar como ejemplo va a ser el minijuego sobre la Casa Batlló.

2.4.5.1. Minijuego Ramblas

Este minijuego consiste en guiar a un taxista a lo largo de la calle de las ramblas evitando todos los obstáculos presentes por el camino. Los actores principales son el taxi y los diferentes coches u obstáculos que este debe esquivar.

`Algoritmos implementados para cada uno de los actores`

- Taxi
  1. El taxi se movera en la dirección y sentido del puntero del ratón.
  2. Si el taxi colisiona con alguno coche u obstáculo el juego finalizara.
  3. El siguiente juego se desbloqueara si el taxi permanece intacto durante 1 minuto.
- Coches u obstáculos
  1. Se generan durante un tiempo aleatorio diferentes coches u obstáculos en cada uno de los diferentes carriles de las ramblas. Estos tendrán una dirección, sentido y velocidad diferentes según el carril donde se encuentren.
  2. Si colisionan con el taxi el juego finaliza.

2.4.5.2. Minijuego Parque Güell

Este minijuego tiene como el objetivo ir recogiendo fragmento de piezas (trencadís) y esquivar otras hasta que el actor principal (salamandra) esté completa. Los actores principales son la salamandra y los objetos a recoger o esquivar.

`Algoritmos implementados para cada uno de los actores`

- Salamandra
    1. La salamandra se movera con las flechas de indicación del teclado.
    2. Si la salamandra colisiona con una de la piezas correctas (trencadís) se rellenara un porcentaje de su estructura.
    3. Si por el contrario la salamandra colisiona con una de las piezas incorrectas se disminuira un porcentaje de su cuerpo.
    4. Para poder desbloquear el siguiente minijuego la salamandra tiene que haber recogido un total de 20 piezas correctas acumuladas, es decir , 20 piezas correctas y ninguna incorrecta. Esto es debido a que un pieza incorrecta descuenta una correcta.
- Piezas

1. Si crearan 2 tipos de piezas en 4 partes distintas de la parte superior de la escena una vez cada 5 segundos , de manera que estas vayan cayendo.
2. Si una pieza colisiona con el suelo se destruye.

2.4.5.3. Minijuego Montjuïc

Este minijuego consiste en ir superando los diferentes peldaños de una montaña hasta llegar a la cima. Los actores principales son los peldaños y el personaje.

`Algoritmos implementados para cada uno de los actores`

- Personaje
    1. El personaje se movera con las flechas de indicación.
    2. Si se pulsa la tecla 'Space' el personaje saltara.
    3. Si el personaje colisiona con el suelo el juego finaliza.
    4. El siguiente minijuego se desbloqueara si el personaje supera 50 peldaños con éxito que son los equivalentes con llegar a la cima.
- Peldaños
    1. Se crearan diferentes peldaños con diferentes propiedades físicas en distintas partes de la escena de manera que el personaje los vaya superando.
    2. Si los peldaños colisionan con el suelo se destruyen.

2.4.5.4. Minijuego Camp Nou

Este minijuego consiste en lanzar una serie de pelotas a la portería intentando que el portero no las pare. Los actores principales son el guante, la porteria y la pelota.

`Algoritmos implementados para cada uno de los actores`

- Pelota
  1. Si se hace click hacia una dirección de la escena, la pelota se mueve hacia esa dirección.
  2. Si colisiona con el guante, el tiro se reinicia.
  3. El siguiente minijuego se desbloquea si se consigue marcar 7 de 10 lanzamientos a puerta.
- Guante
  1. Si colisiona con la pelota, aplica las físicas con ella y se reinicia el lanzamiento.
- Porteria
  1. Si colisiona con la pelota, se aumenta el contador de goles y se reinicia el tiro.

2.4.5.5. Minijuego Sagrada Familia

Este minijuego esta basado en un puzzle de la sagrada familia y por lo tanto tiene como objetivo completarlo. Los actores principales serían las piezas y las casillas.

`Algoritmos implementados para cada uno de los actores`

- Piezas
  1. Cada vez que se hace click en una pieza y no hay ninguna pieza en el aire sus coordenadas son las mismas que las del puntero.
  2. Solo puede haber una pieza en el aire.
  3. Si una pieza esta en el aire y se hace click en un lugar de la escena, la pieza deja de estar en el aire.
- Casillas
  1. Si la pieza correspondiente a cada casilla esta en su lugar correspondiente con un margen de error de pocos milímetros en sus coordenadas, la pieza permanece fijada en esa posición y se aumenta un contador de piezas bien colocadas.
  2. Si todas las piezas estan bien colocadas en sus respectivas casillas en menos de 3 minutos, el juego finaliza y se desbloqueará el siguiente minijuego.

2.4.5.6. Minijuego Casa Batlló

Este es el minijuego que vamos a utilizar como ejemplo para explicar las posibilidades que nos ofrece esta herramienta. Este minijuego consiste en guiar a una paloma a lo largo de un recorrido a traves de la Casa Batlló intentando esquivar las columnas a su paso. Los actores principales son la palomas y las columnas. A continuación mostramos el aspecto inicial de dicho minijuego.

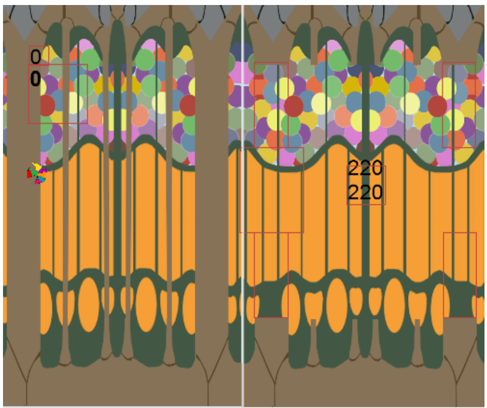

Figura 2.27: Aspecto inicial

`Algoritmos implementados para cada uno de los actores`

- Paloma
  1. Si la paloma colisiona con alguna columna o con el suelo el juego finaliza y devuelve cuantas columnas ha conseguido pasar.
  2. Si la paloma consigue pasar 15 columnas se desbloqueara el siguiente minijuego.
- Columnas
  1. Se generan columnas de un tamaño aleatorio de manera que la paloma pueda pasar por en medio.

A continuación mostramos el diagrama de flujo de cada uno de los algoritmos mencionados anteriormente, con una pequeña explicación de las funcionalidades de los mismos.

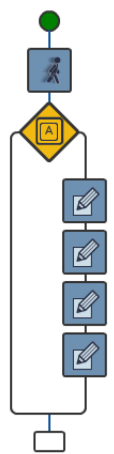

Figura 2.28: Movimiento paloma

Cada vez que se pulse la tecla ”Space”, la paloma tendra una velocidad lineal en y, y un ángulo de 45 grados con respecto a la normal del suelo. Además la paloma es una actor animado con 2 sprites distintos. (ver Figura 2.28)

Figura 2.29: Algoritmo columna superior

Cuando se genera la columna inferior se invoca (spawn) un actor columna en dicha posición con una altura resultante a la resta entre la altura total de la escena y el tamaño de la columna inferior. (ver Figura 2.29)

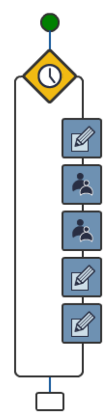

Figura 2.30: Algoritmo columna inferior

Cada cierto tiempo se invoca un actor columna en dicha posición pero con una altura aleatoria, siempre contemplando que haya un espacio mínimo para dejar pasar a la paloma. (ver Figura 2.30)

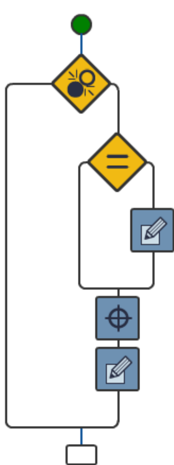

Figura 2.31: Choque paloma

Cada vez que la paloma colisiona con una columna, comprobamos si el número de columnas pasadas es mayor que el récord actual. Si es así se actualiza y se destruye la paloma con el posterior reinicio del minijuego. (ver Figura 2.31)

Capítulo 3

Modelización Matemática de los Objetos

3.1. Introducción y objetivos

Como en la estancia en prácticas he utilizado una herramienta de programación basada en un serie de diagramas de flujo que definen los diferentes comportamientos de los actores en la escena, se ha propuesto realizar una memoria final que pueda contener un aspecto matemático relacionado con la programación de un cierto videojuego.

Se propuso realizar un videojuego en el que se utilizara como objeto un boomerang que, en su movimiento a lo largo de la escena, fuese recogiendo monedas que se acumulan como puntos en el juego. Así que con respecto al aspecto matemático del videojuego, deberia construir un boomerang utilizando bien una interpolación por polinomios de Lagrange o bien una aproximación con la curva de Bézier asociada a determinados polinomios de Bernstein. A continuación se intenta reproducir tanto la trayectoria como el movimiento de este objeto a lo largo de la escena del videojuego.

En este apartado vamos a estudiar la aproximación y representación de curvas cuando estos objetos deben ser procesados por un ordenador, como paso previo o intermedio en el diseño y producción de un objeto con una forma determinada.

Se nos presenta entonces un problema, que es encontrar un método para describir estas curvas lo más adaptado en lo posible a la necesidades y peculiaridades de la programación.

El caso principal lo constituye la transformación de la representación continua dada por una curva en un conjunto discreto de datos. Esto se puede realizar de dos formas, como ya hemos comentado anteriormente:

1. **Mediante la interpolación**, eligiendo un numero finito de puntos de la curva a representar y construyendo una curva polinómica que pasa por dichos puntos. Esta forma es la que se hace usando los polinomios de Lagrange.

2. **Mediante la aproximación**, se elige también un número finito de puntos pero la curva polinómica que se construye no pasa por estos puntos sino que es 'controlada' por ellos. Este segundo método es el que se utiliza para construir las curvas y las superfies de Bézier. Lo que se hace es considerar la curva inscrita en un polígono (si es plana) o en un poliedro si posee torsión) de forma que el polígono o el poliedro la determinan completamente.

En primer lugar cogeremos tres puntos y estudiaremos el polinomio de Lagrange asociado a estos con el objetivo de obtener un polinomio cuya gráfica se asemeje lo más posible a la forma de un boomerang. Una vez hecho esto lo realizaremos 3 veces mas para conseguir una interpolación de un boomerang completa y que posteriormente lo utilizaremos para animarlo. Cabe destacar que estas 4 interpolaciones daran lugar a los diferentes sprites asociados al objeto. Por otra parte realizaremos los mismos pasos pero esta vez utilizando curvas de Bézier (que se construyen utilizando Polinomios de Bernstein) para obtener una aproximación correcta de un boomerang. Por último compararemos ventajas e inconvenientes de utilizar un método frente al otro y analizaremos las diferencias computacionales de crear el sprite de un modo u otro.

3.2. Interpolación con polinomios de Lagrange

3.2.1. Definición de los Polinomios de Lagrange

Defincion 3.2.1.1. Dado un conjunto de $k+1$ puntos $(x_0, y_0), ..., (x_k, y_k)$ donde todos los x_j se asumen distintos, el polinomio interpolador en la forma de Lagrange asociado a $(x_0, y_0), ..., (x_k, y_k)$ es la combinación lineal siguiente:

$$L(x) = \sum_{j=0}^{k} y_j l_j(x) \tag{3.1}$$

donde $l_j(x)$ son las distintos polinomios que forman una base del espacio vectorial $\mathbb{P}_k[x]$ (polinomios de grado $\leq k$) para el conjunto de puntos $(x_0, y_0), ..., (x_k, y_k)$

$$l_j(x) = \prod_{i=0, i\neq j}^{k} \frac{x - x_i}{x_j - x_i} = \frac{x - x_0}{x_j - x_0} ... \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} ... \frac{x - x_k}{x_j - x_k}$$

Observacion 3.2.1.2. $L(x)$ es la formula de lagrange del polinomio de interpolación y $l_j(x)$ son cada uno de los polinomios de lagrange.

3.2.2. Propiedades de los Polinomios de Lagrange

Los polinomios son útiles herramientas matemáticas que además se definen de forma sencilla. Pueden calcularse rápidamente en sistemas computacionales y permiten representar una gran variedad de funciones. Pueden ser derivados e integrados fácilmente y pueden ser unidos para formar curvas que aproximen una función tanto como se desee. Sabemos que un polinomio real de una variable de grado n de la forma

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

puede ser representado como una combinacion lineal de elementos de la base canónica

$$\beta = \{1, x, \ldots, x_n\}$$

del espacio de polinomios $\mathbb{P}_n[x]$. Esta base es tan solo una de un número infinito de bases para el espacio de polinomios. En lo siguiente discutiremos otra de las bases recientemente usadas del espacio de polinomios: los polinomios de Lagrange y analizaremos algunas de sus propiedades

Proposición 3.2.2.1. Sea $\beta = \{l_j(x)\}$ la base $\mathbb{P}_k[x]$ formada por los polinomios de Lagrange asociados a los pares $(x_0, y_0), ..., (x_k, y_k)$ donde $j = 0, \ldots, k$. Entonces:

$$l_j(x_j) = \prod_{i=0, i \neq j}^{k} \frac{x_j - x_i}{x_j - x_i} = 1$$

$$l_j(x_k) = \prod_{i=0, i \neq j}^{k} \frac{x_k - x_i}{x_j - x_i} = 0$$

Demostración.

Es evidente por la definición de $l_j(x)$.

3.2.3. Construcción del sprite mediante Polinomios de Lagrange

Como hemos comentado anteriormente en el Capitulo 1, vamos a usar 4 fórmulas de Lagrange distintas con 3 puntos distintos en cada polinomio interpolador de Lagrange para modelizar los sprites asociados a un boomerang que se esta moviendo. Para cada polinomio interpolador de Lagrange realizaremos los siguientes pasos:

1. En primer lugar obtendremos el polinomio interpolador para cada trío de puntos.
2. A continuación obtendremos la fórmula de Lagrange asociada.
3. Por último mostraremos la gráfica asociada a dicho polinomio.

- 1 Polinomio

  El primer polinomio que se calcula es el polinomio de grado 2 asociado a los puntos (2,0),(0,-2) y (-2,0).

	(x_k, y_k)	$l_j(x_k)$	$l_j(x_k)y_k$
Punto 1	(2,0)	$\frac{x-0}{2-0}\frac{x+2}{2-(-2)} = \frac{x(x+2)}{8}$	0
Punto 2	(0,-2)	$\frac{x-2}{0-2}\frac{x+2}{0-(-2)} = \frac{(x-2)(x+2)}{-4}$	$\frac{-(x-2)(x+2)}{-2}$
Punto 3	(-2,0)	$\frac{x-2}{-2-2}\frac{x}{-2-0} = \frac{(x-2)x}{-8}$	0

Cuadro 3.1: Polinomios de Lagrange asociados al primer Polinomio

Por lo tanto la fórmula de lagrange definida por dichos polinomios es igual a:

$L_1(x) = \frac{-(x-2)(x+2)}{-2}$

Si mostramos la gráfica de dicho polinomio podemos observar el resultado que un principio queriamos obtener:

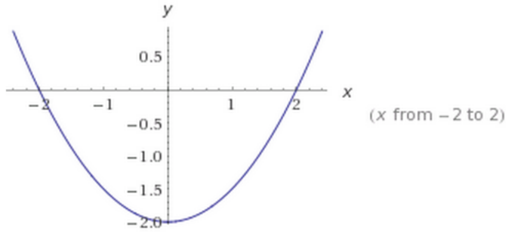

Figura 3.1: Gráfica asociada al primer Polinomio

- 2 Polinomio

  El segundo polinomio que se calcula es el polinomio de grado 2 asociado a los puntos (0,2),(-2,0) y (0,-2), y usamos como variable independiente a la y.

	(x_k, y_k)	$l_j(x_k)$	$l_j(x_k)y_k$
Punto 1	(0,2)	$\frac{y-0}{2-0}\frac{y+2}{2-(-2)} = \frac{y(y+2)}{8}$	0
Punto 2	(-2,0)	$\frac{y-2}{0-2}\frac{y+2}{0-(-2)} = \frac{(y-2)(y+2)}{-4}$	$\frac{(y-2)(y+2)}{2}$
Punto 3	(0,-2)	$\frac{y-2}{-2-2}\frac{y}{-2-0} = \frac{(y-2)y}{8}$	0

Cuadro 3.2: Polinomios de Lagrange asociados al segundo Polinomio

Por lo tanto la fórmula de lagrange definida por dichos polinomios es igual a:
$L_2(y) = \frac{(y-2)(y+2)}{2}$
Si mostramos la gráfica de dicho polinomio podemos observar el resultado que un principio queriamos obtener:

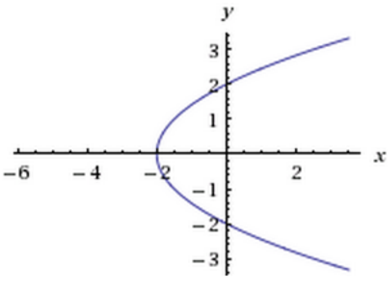

Figura 3.2: Gráfica asociada al segundo Polinomio

- 3 Polinomio

  El tercer polinomio que se calcula es el polinomio de grado 2 asociado a los puntos (2,0),(0,2) y (-2,0).

	(x_k, y_k)	$l_j(x_k)$	$l_j(x_k)y_k$
Punto 1	(2,0)	$\frac{x-0}{2-0}\frac{x+2}{2-(-2)} = \frac{x(x+2)}{8}$	0
Punto 2	(0,2)	$\frac{x-2}{0-2}\frac{x+2}{0-(-2)} = \frac{(x-2)(x+2)}{-4}$	$\frac{-(x-2)(x+2)}{4} \cdot 2$
Punto 3	(-2,0)	$\frac{x-2}{-2-2}\frac{x}{-2-0} = \frac{(x-2)x}{-8}$	0

Cuadro 3.3: Polinomios de Lagrange asociados al tercer Polinomio

Por lo tanto la fórmula de lagrange definida por dichos polinomios es igual a:

$L_3(x) = \frac{-(x-2)(x+2)}{2}$

Si mostramos la gráfica de dicho polinomio podemos observar el resultado que un principio queriamos obtener:

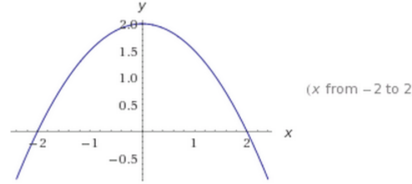

Figura 3.3: Gráfica asociada al tercer Polinomio

- 4 Polinomio

  El cuarto polinomio que se calcula es el polinomio de grado 2 asociado a los puntos (0,2),(2,0) y (0,-2), y usamos la variable y como variable independiente.

	(x_k, y_k)	$l_j(x_k)$	$l_j(x_k)y_k$
Punto 1	(0,2)	$\frac{y-0}{2-0}\frac{y+2}{2-(-2)} = \frac{y(y+2)}{8}$	0
Punto 2	(2,0)	$\frac{y-2}{0-2}\frac{y+2}{0-(-2)} = \frac{(y-2)(y+2)}{-4}$	$\frac{-(y-2)(y+2)}{2}$
Punto 3	(0,-2)	$\frac{y-2}{-2-2}\frac{y}{-2-0} = \frac{(y-2)y}{8}$	0

Cuadro 3.4: Polinomios de Lagrange asociados al cuarto Polinomio

Por lo tanto la fórmula de lagrange definida por dichos polinomios es igual a:

$L_4(y) = \frac{-(y-2)(y+2)}{2}$

Si mostramos la gráfica de dicho polinomio podemos observar el resultado que un principio queriamos obtener:

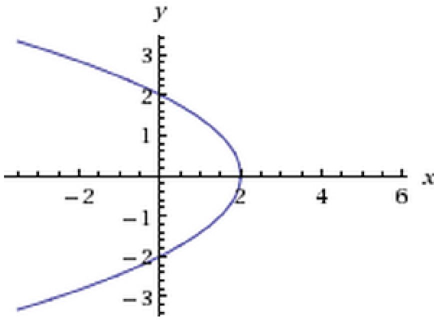

Figura 3.4: Gráfica asociada al cuarto Polinomio

A continuación podemos observar un codigo implementado en MATLAB que nos sirve para construir polinomios de Lagrange asociado a un número de puntos distintos.

Codigo de MATLAB

```
1  function[yi,pol]=lagrange(xs,ys,x)
2  %interpolacion por el polinomio de lagrange
3  %obtener longitud del vector x
4  n=length(xs);
5  %%validar longitud igual
6  if length (ys)~=n, error('xs e ys deben de ser de la misma longitud');end;
7  yi=0;
8  pol='0';
9  %calcular los n factores de lagrange
10
11 for i=1:n
12     %cada factor es el producto de x-xj)/xi-xj) donde i es distinto de j
13     producto=ys(i);
14     termino=num2str(ys(i));
15     for j=1:n
16         if i~=j
17             producto=producto*(x-xs(j))/(xs(i)-xs(j));
18             termino=strcat(termino,'*(x-',num2str(xs(j)),')/(',num2str(xs(i)),'-'
               ,num2str(xs(j)),')');
19         end
20     end
21     %sumar cada termino
22     yi=yi+producto;
23     pol=strcat(pol,'+',termino);
24 end
25 %toolbox
26 pol=sym(pol);
27 pol=simplify(pol);
28 pol=inline(char(pol));
```

3.3. Interpolación con Curvas de Bézier

3.3.1. Introducción

Para construir las curvas planas de Bézier, lo que se hace es considerar la curva como inscrita en un polígono de forma que el polígono la determina completamente.

Hay dos formas de realizar esto:

1. **Algoritmo De Casteljau**, basado en el uso de la interpolación lineal. Es una aproximación conceptual.
2. **Usando los polinomios de Bernstein**, que utilizan una base distribuida del espacio de los polinomios. Es una aproximación más util analíticamente.

Para poder comprender lo definido a continuación, primero deberemos definir el concepto de curva de Bézier y sus propiedades.

3.3.2. Algoritmo de Casteljau

El algoritmo de Casteljau es, en el campo del análisis numérico de la matemática, un método recursivo para calcular curvas de Bézier. Este algoritmo es un método numéricamente estable para evaluar las curvas de Bézier.

La idea principal de este algoritmo surge de requisitos gráficos en informática y se basa en el hecho que una restricción de una curva de Bézier es también una curva de Bézier. Entonces, a partir de la curva inicial se encuentran los puntos de control de dos curvas definidas por $x \in [0, \frac{1}{2}]$ y $x \in [\frac{1}{2}, 1]$ y se fijan los píxeles que corresponden al punto por $x = \frac{1}{2}$, donde se iteran los procesos sobre cada una de las dos curvas hasta que la precisión sea inferior al píxel.

Definición 3.3.2.1 Sean $\{b_0, \ldots, b_n\}$ $n+1$ puntos en $\mathbb{R}^2$. Para cada $x \in [0, 1]$ consideramos la siguiente recurrencia:

$$\begin{aligned} b_i^0 &:= b_i, \quad i = 0, \ldots, n \\ b_i^r &:= b_i^{r-1}(1-x) + b_{i+1}^{r-1}x, \quad i = 0, \ldots, n-r \ , \ r = 1, \ldots n \end{aligned} \tag{3.2}$$

Entonces se define la curva de Bezier de grado n asociada a $b_0 \ldots b_n$ como la curva parametrizada siguiente.

Definicion 3.3.2.2 Dados los puntos de control $b_0, b_1, ..., b_n$ podemos definir una curva de Bezier mediante el Algoritmo de Casteljau:

$$\begin{aligned} \alpha : [0,1] &\longrightarrow \mathbb{R}^2 \\ x &\longmapsto \alpha(x) := b_0^n(x) \end{aligned} \tag{3.3}$$

siendo ${b_0}^n(x)$ obtenido mediante la recurrencia definida en 3.3.2.1 y se denota como

$$\alpha(x) \equiv B[b_0, \ldots, b_n](x).\forall x \in [0,1] \tag{3.4}$$

Nota: A continuación podemos ver la iteracion con más detalle. Fijamos $x \in [0,1]$

- Para r= 0

  $b_0^0(x) = b_0$

  $b_1^0(x) = b_1$

  .

  .

  .

  $b_n^0(x) = b_n$

- Para r=1

  $b^1(x) = (1-x)b_0^0(x) + xb_1^0(x)$

  $b_1^1(x) = (1-x)b_1^0(x) + xb_2^0(x)$

  $b_2^1(x) = (1-x)b_2^0(x) + xb_3^0(x)$

  .

  .

  .

  $b_{n-1}^1(x) = (1-x)b_{n-1}^0(x) + xb_n^0(x)$

- Para r=2

  $b_0^2(x) = (1-x)b_0^1(x) + xb_1^1(x)$

  $b_1^2(x) = (1-x)b_1^1(x) + xb_2^1(x)$

  $b_2^2(x) = (1-x)b_2^1(x) + xb_3^1(x)$

  .

  .

  .

  $b_{n-2}^2(x) = (1-x)b_{n-2}^1(x) + xb_{n-1}^1(x)$

- Y asi sucesivamente hasta llegar a r=n
  $b_0^n(x) = (1-x)b_0^{n-1}(x) + xb_1^{n-1}(x)$

Gráficamente lo podemos observar en el siguente gráfico:

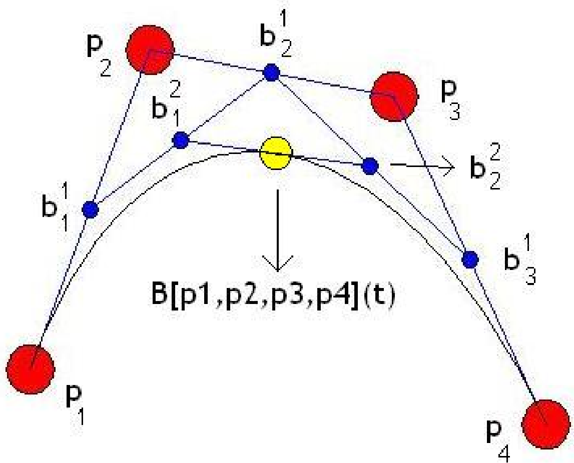

Figura 3.5: Algoritmo de Casteljau para 4 puntos de control

El Algoritmo de Casteljau nos permite inferir un par de propiedades importantes de las curvas de Bezier. No obstante debemos definir en primer lugar los siguientes conceptos:

Definición 3.3.2.3. Una combinación baricéntrica se define como una combinación en la que la suma de sus puntos es igual a 1.

$$b = \sum_{j=0}^{n} \alpha_j b_j; \tag{3.5}$$

con $b_j \in \mathbb{R}^2$ y $\alpha_0 + \ldots + \alpha_n = 1$

Definición 3.3.2.4. Una aplicación Φ se llama aplicación afín si deja invariante a las combinaciones baricentricas. Así que si

$$x = \sum \alpha_j a_j; \quad x, a_j \in \mathbb{R}^2 \tag{3.6}$$

y Φ es una aplicacion afin, se cumple que:

$$\Phi(x) = \sum \alpha_j \Phi(a_j); \quad \Phi(x), \Phi(a_j) \in \mathbb{R}^2 \tag{3.7}$$

Definición 3.3.2.5. P es una combinación convexa de $p_0, \ldots, p_n$ sii $\exists \alpha_0, \ldots, \alpha_n \in \mathbb{R}$ tales que

$$\begin{cases} (1) & 0 \leq \alpha_i \leq 1 \qquad \text{Para todo } i = 0, \ldots, n \\ (2) & \text{p} = \sum_{i=0}^{n} \alpha_i p_i \end{cases}$$

Definición 3.3.2.6. Envoltura convexa de un conjunto de puntos $\{p_0, \ldots, p_n\}$ en $\mathbb{R}^2$

El conjunto de todos los puntos p que pueden escribirse como combinacion convexa de los puntos $p_0, p_1, \ldots, p_n$ es llamado envoltura convexa de $p_0, p_1, \ldots, p_n$. Esta envoltura convexa es el conjunto convexo más pequeño que contiene al conjunto de puntos $p_0, p_1, \ldots, p_n$ y se denota como $Conv(p_0, \ldots, p_n)$.

Una vez visto lo anterior pasemos a definir dos propiedades fundamentales de las Curvas de Bezier. Más adelante estudiaremos otras propiedades.

Proposición 3.3.2.7. Propiedades

1. **Invarianza Afín.**

   Sea la curva de Bezier $\alpha \equiv B[b_0, \ldots, b_n]$.

   Sea $\Phi : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ una aplicacion afín. Entonces:

$$\Phi(\alpha) = \Phi(B[b_0, \ldots, b_n]) = B[\Phi(b_0), \ldots, \Phi(b_n)] \tag{3.8}$$

2. **Propiedad de la envoltura convexa.**

   Para todo $x \in [0, 1]$

$$\alpha \equiv B[b_0, \ldots, b_n](x) \in Conv(b_0, \ldots, b_n) \tag{3.9}$$

Demostración

1.Tomando la primera propiedad:

$$\alpha \equiv B[b_0, \ldots, b_n] = b_0^n = (1-x)b_0^{n-1}(x) + xb_1^{n-1}(x) \tag{3.10}$$

Entonces, como Φ es afín

$$\Phi(\alpha) = (1-x)\Phi(b_0^{n-1}(x)) + x\Phi(b_1^{n-1}(x)) \tag{3.11}$$

Razonando sucesivamente se obtiene:

$$\Phi(\alpha) = B[\Phi(b_0), \ldots, \Phi(b_n)] \tag{3.12}$$

porque los distintos $b_i^r(x)$ son combinaciones baricéntricas.

2. Tomando la segunda propiedad obtenemos:

$$\alpha(x) = B[b_0, \ldots, b_n](x) = b_0^n(x) = (1-x)b_0^{n-1}(x) + xb_1^{n-1}(x) \tag{3.13}$$

Retomando sobre la recursión

$$\alpha(x) = Conv(\{b_0, \ldots, b_n\}) \tag{3.14}$$

es decir, son combinaciones convexas recursivas que involucran recursivamente a todos los puntos $\{b_0, \ldots, b_n\}$.

Hay otra forma de ver las curvas de Bézier, y es expresando estas en función de los polinomios de Bernstein.

3.3.3. Polinomios de Bernstein

Básicamente los polinomios de Bernstein permiten aproximar una función continua f definida en un intérvalo cerrado y acotado $[a, b]$, , asi como ajustar curvas o superficies. También permiten aproximar un conjunto de datos (como por ejemplo funciones poligonales). En este último caso, por aproximación entenderemos una función que aproxima a a la funcion verdadera pero no necesariamente reproduce el conjunto de datos exactamente. Esto es, la gráfica de la función que aproxima no pasara a traves de los datos sino cerca de ellos. Como herramienta de ajuste, estos polinomios tienen varias ventajas, principalmente en diseño asistido por ordenador. Existen circunstancias o experimentos en los que conviene encontrar o quedarse con una buena aproximación mas que buscar interpolar los datos.

Los polinomios de Bernstein son llamados así porque son la base primordial de la demostración que Bernstein realizo del Teorema de aproximacion de Weierstrass. La esencia de esta prueba es la construcción de una sucesión de polinomios que, como veremos en este capítulo, convergen uniformemente a una funcion continua. La demostración que realizo Bernstein aparecio en 1912 y es de las ultimas pruebas que se dieron del teorema de Weierstrass. Dicha demostración ha tenido un profundo impacto en muchas áreas. Los polinomios de Bernstein estan conectados con la teoria de Probabilidad, con problemas sobre momentos, con la teoria de sumas de series divergentes. Problemas interesantes en análisis complejo, algunos de los cuales no han sido aún completamente resueltos, se refieren al comportamiento de los polinomios de Bernstein de funciones analíticas.

3.3.3.1. Definición y ejemplos

Definición 1. Los polinomios de Bernstein se definen como

$$B_i^n(x) = \binom{n}{i} x^i (1-x)^{n-i} \tag{3.15}$$

con $i = 0, 1, \ldots, n$ y $x \in [0, 1]$.

Observación 1. Por conveniencia diremos que $B_i^n = 0$ si $i < 0$ ó $i > n$.

Definición 2. Sea $f : [0,1] \longrightarrow \mathbb{R}$ una función continua. Se define el polinomio aproximante de Berstein de grado n como:

$$B_n(f;x) = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} f(\frac{i}{n}) = \sum_{i=0}^{n} B_n^i(x) f(\frac{i}{n}) \tag{3.16}$$

Observación 2. El polinomio de Bernstein de grado n asociado a la funcion $f : [a,b] \subset \mathbb{R} \longrightarrow \mathbb{R}$ continua esta definido de la siguiente forma:

$$B_n(f;x) := B_n(f \circ T, t) = \sum_{i=0}^{n} \binom{n}{i} (t-a)^i (b-t)^{n-i} f \circ T(\frac{i}{n}) \tag{3.17}$$

donde $T : [0,1] \longrightarrow [a,b]$ se define como $T(t) = (1-t)a + tb = x$. Así $f \circ T$ es continua.

Para quienes esten familiarizados con la Teoria de Probabilidades, reconoceran a los polinomios básicos de Bernstein como las funciones de densidad para una distribucion binomial. Especificamente, $B_i^n(x)$ es la probabilidad de lograr exactamente i exitos en una sucesion de n pruebas o ensayos independientes, en el cual la probabilidad de exito en cualquier prueba es x.

Ejemplo 1.

1) Polinomios básicos de Bernstein de grado 1

$$B_0^1(x) = \binom{n}{0} x^0 (1-x)^1 = 1 - x$$
$$B_1^1(x) = x$$

con $0 \leq x \leq 1$.

2) Polinomios básicos de Bernstein de grado 2

$$B_0^2(x) = (1-x)^2$$
$$B_1^2(x) = 2x(1-x)$$
$$B_2^2(x) = x^2$$

3) Polinomios básicos de Bernstein de grado 3

$$\begin{aligned} B_0^3(x) &= (1-x)^3 \\ B_1^3(x) &= 3x(1-x)^2 \\ B_2^3(x) &= 3x^2(1-x) \\ B_3^3(x) &= x^3 \end{aligned}$$

Proposicion 1. El conjunto de los polinomios de Bernstein de grado n, $\{B_i^n(x)\}_{i=0}^n$, es una base de $\mathbb{P}_n[x]$

Demostración. Vamos a demostrar en primer lugar que, para cada $n > 0$ y $i \in \{0, \ldots, n\}$, tenemos:

$$B_i^n(x) = \sum_{k=0}^{n-1} (-1)^k \binom{n}{i} \binom{n-i}{k} x^{k+i} \tag{3.18}$$

Para ello, se tiene que:

$$B_i^n(x) = \binom{n}{i} x^i (1-x)^{n-i} \tag{3.19}$$

Utilizando el binomio de newton para desarrollar $(1-x)^{n-i}$ se tiene que:

$$(1-x)^{n-i} = \sum_{k=0}^{n-i} \binom{n-i}{k} (-x)^{n-i-k} \tag{3.20}$$

Entonces, si seguimos a partir de (3.19):

$$\begin{aligned} \binom{n}{i} x^i (1-x)^{n-i} &= \binom{n}{i} x^i \sum_{k=0}^{n-i} (-x)^{n-i-k} \\ &= \sum_{k=0}^{n-i} (-1)^{n-i-k} \binom{n}{i} \binom{n-i}{k} x^{n-k} \end{aligned} \tag{3.21}$$

y el resultado se obtiene después de substituir el índice k en el sumatorio por $n - i - k$ y recordando que $\binom{n-i}{n-i-k} = \binom{n-i}{k}$.

Entonces la matriz que relaciona las $B_i^n(x)$ y los elementos de la base canónica x^i, es triangular superior y los elementos de la diagonal son $\binom{n}{i} i = 0, \ldots, n$.

Tenemos la expresión:

$$\begin{bmatrix} B_0^n(x) \\ \vdots \\ B_n^n(x) \end{bmatrix} = \begin{bmatrix} 1 & \dots & (-1)^n \\ \vdots & \binom{n}{i} & \vdots \\ 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ x^n \end{bmatrix}$$

Como el determinante de la expresion anterior es distinto de 0, los $\{B_i^n(x)\}_{i=0}^n$ son linealmente independientes.

Proposición 2. Vamos a expresar ahora a los polinomios de la base canónica en función de los polinomios de Bernstein

$$x^i = \sum_{k=0}^{n-1} \frac{\binom{k+i}{i}}{\binom{n}{i}} B_{k+1}^n(x) \tag{3.22}$$

Demostración. Operando a partir del miembro de la izquierda y teniendo en cuenta que $\sum_{k=0}^{n-i} B_k^{n-i}(x) = 1 \;\; \forall x \in [0,1]$ en virtud de la Proposición 1 de 3.3.3.2:

$$\begin{aligned} &\sum_{k=0}^{n-1} \frac{\binom{k+i}{i}}{\binom{n}{i}} B_{k+1}^n(x) \\ &= \sum_{=k=0}^{n-1} \frac{\binom{k+i}{i}}{\binom{n}{i}} \sum_{l=0}^{n-i-k} (-1)^l \binom{n}{i+k} \binom{n-i-k}{l} x^{l+k+i} \\ &= x^i \sum_{k=0}^{n-i} \frac{\frac{(k+i)!}{i!k!}}{\frac{n!}{i!(n-i)!}} x^k \sum_{l=0}^{n-i-k} (-1)^l \frac{n!}{(i+k)!(n-i-k)!} \binom{n-i-k}{l} x^l \\ &= x^i \sum_{k=0}^{n-i} \binom{n-i}{k} x^k \sum_{l=0}^{n-i-k} \binom{n-i-k}{l} (-x)^l \\ &= x^i \sum_{k=0}^{n-i} \binom{n-i}{k} x^k (1-x)^{n-i-k} \\ . &= x^i \sum_{k=0}^{n-i} B_k^{n-i} = x^i \end{aligned} \tag{3.23}$$

Proposicion 3. Los polinomios de Bernstein $B_i^n(x)$ tienen un único máximo en $x = \frac{i}{n}$

Demostración. Primero, calculemos los puntos críticos de $B_i^n(x)$. Si derivamos $B_i^n(x)$ e igualamos a 0 obtenemos:

$$x^{i-1}(1-x)^{n-i}(i(1-x) - (n-i)x) = 0, \tag{3.24}$$

o equivalentemente

$$x^{i-1}(1-x)^{n-i}(i - nx) = 0, \tag{3.25}$$

Por tanto, los puntos críticos se encuentran en $x = 0, x = 1$ o en $x = \frac{i}{n}$. Debido a la propiedad de las condiciones de los extremos y a la no negatividad de los polinomios de Bernstein, el punto crítico $x = \frac{i}{n}$ es el único máximo local.

3.3.3.2. Propiedades de los Polinomios de Bernstein

A continuación mostraremos algunas propiedades de los polinomios básicos de Bernstein.

Proposicion 1. Propiedades de los Polinomios de Bernstein

En el intervalo [0,1], los polinomios de Bernstein de grado n satisfacen:

1. $0 \leq B_k^n(x) \leq 1$ para todo $x \in [0,1], k = 0, \ldots, n$
2. $\sum_{k=0}^{n} B_k^n = 1$ para todo $x \in [0,1]$
3. $B_k^n(x) = (1-x)B_k^{n-1}(x) + xB_{k-1}^{n-1}(x)$ *Recurrencia*
4. $B_k^n(x) = B_{n-k}^n(1-x)$ *Simetría*
5. $\sum_{i=0}^{n} B_i^n(x) = \sum_{i=0}^{n-1} B_i^{n-1}(x)$
6. $\sum_{i=0}^{n} \frac{i}{n} B_i^n(x) = x$ *Precisión Lineal*

Demostración.

1. Vamos a razonar por inducción sobre el grado de los polinomios. Entonces:

i) $n = 1$. En este caso es inmediato que los polinomios basicos de Bernstein $B_0^1(x) = (1-x)$ y $B_1^1(x) = x$ son no negativos si $0 \leq x \leq 1$.

ii) Supongamos que todos los polinomios basicos de Bernstein de grado $m < n$ son no negativos. Luego, a partir de la definición recursiva tomando $m = n$

$$B_i^m(x) = (1-x)B_i^{m-1}(x) + xB_{i-1}^{m-1}(x) \tag{3.26}$$

Entonces $B_i^m(x) \geq 0$ para $0 \leq x \leq 1$, ya que por hipótesis de inducción los elementos del lado derecho de esta igualdad son no negativos.

2. Usando el desarrollo binomial, tenemos que:

$$1 = (x + (1-x))^n = \sum_{i=0}^{n} \binom{n}{i} x^i (1-x)^{n-i} \tag{3.27}$$

3. Desarrollando a partir del segundo miembro:

$$\begin{aligned}
&(1-x)B_k^{n-1}(x) + xB_{k-1}^{n-1}(x) = \\
&(1-x)\binom{n-1}{k}x^k(1-x)^{n-1-k} + x\binom{n-1}{k-1}x^k - 1(1-x)^{n-1-(k-1)} \\
&= \binom{n-1}{k}x^k(1-x)^{n-k} + \binom{n-1}{k-1}x^k(1-x)^{n-k} \\
&= [\binom{n-1}{k} + \binom{n-1}{k-1}]x^k(1-x)^{n-k} \\
&= \binom{n}{k}x^k(1-x)^{n-k} \\
&= B_k^n(x)
\end{aligned} \tag{3.28}$$

4. Usando la definición:

$$B_{n-k}^n(1-x) = \binom{n}{n-k}(1-x)^{n-k}(1-(1-x))^{n-(n-k)} = \binom{n}{k}x^k(1-x)^{n-k} = B_k^n(x) \tag{3.29}$$

5. Como $B_i^n(x) = \binom{n}{i}x^i(1-x)^{n-i}$, $i = 0, \ldots, n$ y $B_i^{n-1}(x) = \binom{n-1}{i}x^i(1-x)^{n-1-i}$, $i = 0, \ldots, n-1$ la propiedad de recurrencia nos dice lo siguiente:

$$\begin{aligned}
\sum_{i=0}^{n} B_i^n(x) &= \sum_{i=0}^{n}[(1-x)B_i^{n-1}(x) + xB_{i-1}^{n-1}(x)] \\
&= (1-x)\sum_{i=0}^{n-1} B_i^{n-1}(x) + x\sum_{i=1}^{n} B_{i-1}^{n-1}(x) \\
&= (1-x)\sum_{i=0}^{n-1} B_i^{n-1}(x) + x\sum_{i=0}^{n-1} B_i^{n-1}(x) \\
&= \sum_{i=0}^{n-1} B_i^{n-1}(x)
\end{aligned} \tag{3.30}$$

6. Cuando i=1 en la ecuación (3.22), tenemos:

$$x = \sum_{k=0}^{n-1} \frac{\binom{k+1}{1}}{\binom{n}{1}} B_{k+1}^n(x) = \sum_{k=0}^{n-1} \frac{k+1}{n} B_{k+1}^n(x) \tag{3.31}$$

Si tomamos $j = k+1$ entonces:

$$\sum_{k=0}^{n-1} \frac{k+1}{n} B_{k+1}^n(x) = \sum_{j=1}^{n-1} \frac{j}{n} B_j^n(x) = \tag{3.32}$$

y, como $\frac{0}{n}B_0^n(x) = 0$ para todo x entonces:

$$x = \sum_{j=i}^{n-1} \frac{j}{n} B_j^n(x) = \sum_{j=0}^{n-1} \frac{j}{n} B_j^n(x) \tag{3.33}$$

3.3.3.3. Un ejemplo de Aproximación a una función

Construir el polinomio de Bernstein que aproxima a $f(x) = \sin 2xconx \subset [0, 1]$.

Solución. Para una aproximacion de primer grado tenemos:

$$B_0^1(x) = 1 - x,\ B_1^1(x) = x$$

Evaluando en los extremos del intervalo:

$$\sin 0 = 0 \text{ y } \sin 2 = 0{,}9093$$

Tenemos entonces

$$B(\sin 2x; x) = \sum_{i=0}^{1} f(\frac{i}{1}) B_i^1(x) = (\sin 2 \cdot 0)(1 - x) + (\sin 2 \cdot 1)x = 0{,}9093x$$

que es una recta que interpola a la funcion $\sin 2x$ en los puntos 0 y 0.9093. Para una aproximación de segundo grado usamos los polinomios básicos

$$B_0^2(x) = (1 - x)^2\ ,\ B_1^2(x) = 2x(1 - x),\ B_2^2(x) = x^2$$

y los valores son

$$\sin 2 \cdot 0 = 0, \sin 2(1/2) = 0{,}8415, \sin 2 \cdot 1 = 0{,}9093$$

Asi la aproximación sera

$$B(\sin 2x; x) = \sum_{i=0}^{2} f(\frac{i}{2}) B_i^2(x)$$

$$= \sin 2 \cdot 0(1 - x)2 + (\sin 1)2x(1 - x) + (\sin 2)x^2 = 1{,}6830x(1 - x) + 0{,}9093x^2$$

que nuevamente interpola a la funcion original en los dos puntos finales.

3.3.3.4. Polinomios de Bersntein y Curvas de Bezier

Teorema 3.3.3.4.1. . Sea $\alpha = B[b_0, \ldots, b_n]$ curva de Bezier asociada a $b_0, \ldots, b_n$. Entonces, para todo $x \in [0, 1]$,

$$\alpha(x) = \sum_{i=0}^{n} B_i^n(x) b_i \tag{3.34}$$

Demostración Sean los puntos $b_0, \ldots, b_n \in \mathbb{R}^n$, vamos a aplicar el algoritmo de Casteljau.

1. En la iteración 0-ésima:
   $b_0^0(x) = b_0$
   $b_1^0(x) = b_1$
   .
   .
   .
   $b_n^0(x) = b_n$
   Por lo tanto:
   $b_i(x) = (b_i) = B_0^0(x) b_i. \ \forall i = 0, \ldots, n$
2. En la iteración 1-ésima:
   $b_0^1(x) = (1 - x) b_0^0 + x b_1^0 = B_0^1(x) b_0^0 + B_1^1(x) b_1^0$
   $b_1^1(x) = (1 - x) b_1^0 + x b_2^0 = B_0^1(x) b_1^0 + B_1^1(x) b_2^0$
   .
   .
   .
   Por tanto:
   $b_i^1(x) = B_0^1(x) b_i^0 + B_1^1(x) b_{i+1}^0. \ \forall i = 0, \ldots, n - 1$
3. En la iteración 2-ésima:
   $b_i^2(x) = (1 - x) b_i^1 + x b_{i+1}^1$
   $= (1 - x) B_0^1(x) b_i + (1 - x) B_1^1(x) + x B_0^1 b_{i+1} + x B_1^1(x) b_{i+2}$
   $= B_0^2(x) b_i + B_1^2(x) b_{i+1} + B_2^2(x) b_{i+2}^0. \ \forall i = 0, \ldots, n - 2$
4. Asi en la iteración r-ésima:
   $b_i^r(x) = B_0^r(x) b_i + B_1^r(x) b_{i+1} + \ldots + B_r^r(x) b_{i+r}. \ \forall i = 0, \ldots, n - r$
5. Y finalmente , para r=n
   $b_0^n(x) = \sum_{i=0}^{n} B_i^n(x) b_i$

Como hemos comentado anteriormente hay un numero infinito de bases para el espacio de polinomios y a continuación vamos a discutir otra de estas bases: los polinomios de Bernstein y analizaremos algunas de sus propiedades.

3.3.4. Propiedades de las Curvas de Bezier

Volvemos a estudiar las propiedades de las curvas de Bezier a la luz de su expresión en función de los polinomios de Bersntein ampliando su listado.

1. Invarianza frente a transformaciones afines del parámetro

Podemos definir los polinomios de Bernstein en el intervalo $[a, b]$ mediante la transformación

$$\begin{aligned} F : [a, b] &\longrightarrow [0, 1] \\ u &\longmapsto F(u) = \frac{u-a}{b-a} = x \end{aligned} \tag{3.35}$$

Entonces

$$B_k^n(u) = B_k^n(x) \tag{3.36}$$

Algebraicamente esta propiedad se sigue de:

$$\sum_{i=0}^{n} b_i B_i^n(x) = \sum_{i=0}^{n} b_i B_i^n(\frac{u-a}{b-a}) \tag{3.37}$$

2. Interpolación en los puntos inicial y final

Vamos a ver que $\alpha(0) = b_0$ y $\alpha(1) = b_n$

$\alpha(0) = \sum_{i=0}^{n} B_i^n(0) b_i = b_0$ ya que:

$$B_i^n(x) = \binom{n}{i} x^i (1-x)^{n-i} \tag{3.38}$$

con:

$$\mathrm{B}_i^n(0) = \begin{cases} 1, \text{ si} & \text{i} \in \{1, \dots, n\} \\ 0, \text{ si} & \text{i=0} \end{cases}$$

Por otra parte,

$\alpha(1) = \sum_{i=0}^{n} B_i^n(1)b_i = b_n$ ya que:

$$B_i^n(x) = \binom{n}{i} x^i (1-x)^{n-i} \tag{3.39}$$

con:

$$\mathrm{B}_i^n(1) = \begin{cases} 1, \text{ si } & \text{i} \in \{1, \ldots, n\} \\ 0, \text{ si } & \text{i=0} \end{cases}$$

3. Simetría

$$\begin{aligned} \alpha(x) &= \sum_{i=0}^{n} B_i^n(x)b_i = \sum_{i=0}^{n} B_{n-i}^n(1-x)b_i = \sum_{i=0}^{n} B_{n-i}^n(1-x)a_{n-i} \\ &= \sum_{j=0}^{n} B_j^n(1-x)a_j = \sum_{j=0}^{n} B_j^n(s)a_j = \beta(x) \end{aligned} \tag{3.40}$$

donde $b_i = a_{n-i}$, $j = n - i$ y $s = 1 - x$

Notar que $\beta(x)$ es la curva de Bézier con puntos de control $a_0 = b_n, a_1 = b_{n-1}, \ldots, a_n = b_0$ que se recorre en sentido inverso a $\alpha(x)$.

4. Precisión lineal

Si $b_0, \ldots, b_n$ son puntos equidistantes en una recta(recta que pasa por b_0 y $b_n : \gamma(x) = xb_n + (1-x)b_0 \;\; x \in [0,1]$. Ademas $b_i = \gamma(\frac{i}{n}) = \frac{i}{n}b_n + (1 - \frac{i}{n})b_0)$. Entonces, como sabemos que $x = \sum_{i=0}^{n} B_i^n(x)\frac{i}{n}$ se tiene:

$$\begin{aligned} \alpha(x) &= \sum_{i=0}^{n} B_i^n(x)b_i = \sum_{i=0}^{n} B_i^n(x)[\frac{i}{n}b_n + (1 - \frac{i}{n})b_0)] \\ &= b_n \sum_{i=0}^{n} B_i^n(x)\frac{i}{n} + b_0 \sum_{i=0}^{n} B_i^n(x) - b_0 \sum_{i=0}^{n} B_i^n(x)\frac{i}{n} \\ &= (b_n - b_0) \sum_{i=0}^{n} B_i^n(x)\frac{i}{n} + b_0 = (b_n - b_0)x + b_0 = xb_n + (1-x)b_0 = \gamma(x) \end{aligned} \tag{3.41}$$

5. Control pseudo-local

Los polinomios de Bernstein $B_i^n(x)$ solo tienen un m,aximo cuando $x = \frac{i}{n}$. Si α y β son curvas de Bézier cuyos polígonos de control difieren en los puntos b_i y $\widetilde{b_i}$ entonces:

$$\begin{aligned} \alpha(x) - \beta(x) &= \alpha[b_0, \ldots, b_n](x) - \beta[\widetilde{b_0}, \ldots, \widetilde{b_n}(x) \\ &= \sum_{j=0}^{n} B_j^n(x)(b_i - \widetilde{b_i}) = B_i^n(x)(b_i - \widetilde{b_i}) \end{aligned} \tag{3.42}$$

y como el máximo de $B_i^n(x)$ es $x = \frac{i}{n}$, es alrededor de $x = \frac{i}{n}$ donde el cambio es más notable.

3.3.5. Construcción del sprite mediante Curvas de Bézier

Al igual que hemos hecho con los Polinomios de Lagrange, vamos a calcular 4 Curvas de Bézier distintas con 3 puntos diferentes en cada una, para modelizar los sprites asociados a un boomerang que se esta moviendo, pero en este caso utilizando Polinomios de Bernstein.

En primer lugar vamos a destacar que, para realizar la aproximación de un boomerang, vamos a utilizar los polinomios de Bernstein de grado 2. Cómo ya hemos calculado anteriormente en el ejemplo 1 del apartado 3.3.3.1, estos polinomios son:

$$\begin{aligned} B_0^2(x) &= (1-x)^2 \\ B_1^2(x) &= 2x(1-x) \\ B_2^2(x) &= x^2 \end{aligned}$$

Una vez visto esto, nos disponemos a realizar las diferentes aproximaciones de las diferentes posiciones del boomerang(sprites).

- 1 Curva

  La primera curva de Bézier que se calcula está asociada a los puntos $b_0 = (-1, 1)$, $b_1 = (0, -1)$ $b_2 = (1, 1)$. Siguiendo (3.34) a partir de estos puntos obtenemos:

$$\begin{aligned} \alpha(x) &= \sum_{i=0}^{2} B_i^2(x) b_i = B_0^2(x) b_0 + B_1^2(x) b_1 + B_2^2(x) b_2 \\ &= (1-x)^2(-1,1) + 2x(1-x)(0,-1) + x^2(1,1) \\ &= (-(1-x)^2 + x^2, (1-x)^2 - 2x(1-x) + x^2) \\ &= (-1+2x, 1-4x+4x^2) \end{aligned}$$

Reparametrizando obtenemos que $\alpha(s) = (s, s^2)$ con $s = -1 + 2x$.

Si mostramos la gráfica de dicha curva de Bézier podemos observar el resultado que un principio queriamos obtener:

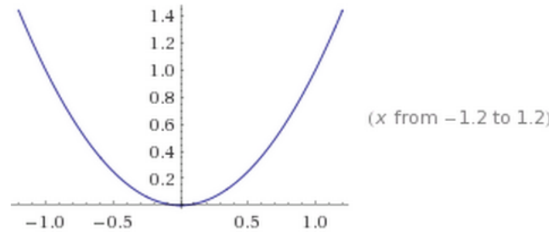

Figura 3.6: Gráfica asociada a la primera curva de Bézier

- 2 Curva

  La segunda curva de Bézier que se calcula está asociada a los puntos $b_0 = (1, 1)$, $b_1 = (-1, 0)$ $b_2 = (1, -1)$. Siguiendo (3.34) a partir de estos puntos obtenemos:

$$\begin{aligned}
\alpha(x) &= \sum_{i=0}^{2} B_i^2(x) b_i = B_0^2(x) b_0 + B_1^2(x) b_1 + B_2^2(x) b_2 \\
&= (1-x)^2(1,1) + 2x(1-x)(-1,0) + x^2(1,-1) \\
&= ((1-x)^2 - 2x(1-x) + x^2, (1-x)^2 - x^2) \\
&= (1 - 4x + 4x^2, 1 - 2x)
\end{aligned}$$

Reparametrizando obtenemos que $\alpha(s) = (s^2, s)$ con $s = 1 - 2x$.

Si mostramos la gráfica de dicha curva de Bézier podemos observar el resultado que un principio queriamos obtener:

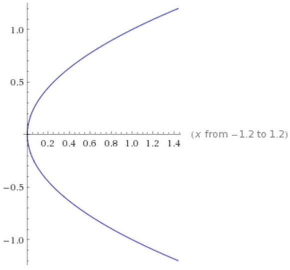

Figura 3.7: Gráfica asociada a la segunda curva de Bézier

- 3 Curva

  La tercera curva de Bézier que se calcula está asociada a los puntos $b_0 = (-1, -1)$, $b_1 = (0, 1)$ $b_2 = (1, -1)$. Siguiendo (3.34) a partir de estos puntos obtenemos:

$$\begin{aligned}\alpha(x) &= \sum_{i=0}^{2} B_i^2(x)b_i = B_0^2(x)b_0 + B_1^2(x)b_1 + B_2^2(x)b_2 \\ &= (1-x)^2(-1,-1) + 2x(1-x)(0,1) + x^2(1,-1) \\ &= (-(1-x)^2 + x^2, -(1-x)^2 + 2x(1-x) - x^2) \\ &= (-1+2x, -1+4x-4x^2)\end{aligned}$$

Reparametrizando obtenemos que $\alpha(s) = (s, -s^2)$ con $s = -1 + 2x$.

Si mostramos la gráfica de dicha curva de Bézier podemos observar el resultado que un principio queriamos obtener:

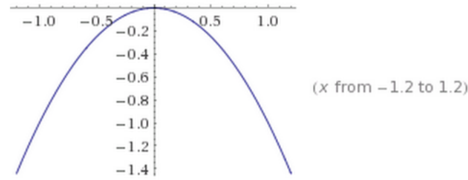

Figura 3.8: Gráfica asociada a la tercera curva de Bézier

- 4 Curva

  La cuarta curva de Bézier que se calcula está asociada a los puntos $b_0 = (-1, 1)$, $b_1 = (1, 0)$ $b_2 = (-1, -1)$. Siguiendo (3.34) a partir de estos puntos obtenemos:

$$\begin{aligned}\alpha(x) &= \sum_{i=0}^{2} B_i^2(x)b_i = B_0^2(x)b_0 + B_1^2(x)b_1 + B_2^2(x)b_2 \\ &= (1-x)^2(-1,1) + 2x(1-x)(1,0) + x^2(-1,-1) \\ &= (-(1-x)^2 + 2x(1-x) - x^2, (1-x)^2 - x^2) \\ &= (-1+4x-4x^2, 1-2x)\end{aligned}$$

Reparametrizando obtenemos que $\alpha(s) = (-s^2, s)$ con $s = 1 - 2x$.

Si mostramos la gráfica de dicha curva de Bézier podemos observar el resultado que un principio queriamos obtener:

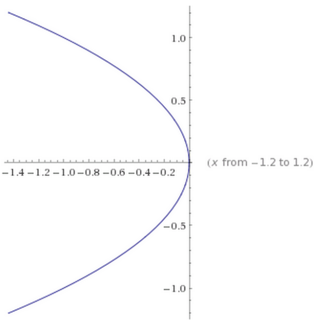

Figura 3.9: Gráfica asociada a la cuarta curva de Bézier

En último lugar veremos el pseudocódigo asociado a la implementación de los polinomios de Bernstein y posteriormente veremos el código utilizado en JAVA para su implementación.

Algoritmo de Evaluación de los polinomios básicos de Bernstein.

```
 Entrada: n, x

 Inicializar: B_{0} = 1

Calcular:

para i = 1 hasta n hacer

B_{i} := x B_{i-1}

 para j = i -1 hasta 1

B_{j} := (1-x)Bj + x -Bj- 1

        B_{0} := (1-x) -B_{0}

        Salida:

para i = 0 hasta n

B_{n,i}(x) := B_{i}
```

Mediante el código implementado a continuación hemos generado las diferentes curvas de Bézier asociadas a nuestros puntos.

CODIGO INTERFAZ GRÁFICA EN JAVA

```
1  import  java.awt.*;
2  import java.awt.event.*;
3  public class Bezier extends Frame implements ItemListener{
4  private Panel panel;
5  private Panel panelc;
6  private Panel panel2;
7
8  private MyCanvas canvas;
9  private Button btnLimpiar;
10 private Checkbox calidad;
11 private Checkbox desliz;
12 private Checkbox pol;
13
14 private TextField px;
15 private TextField py;
16 private Button btnAnadir;
17
18 private static TextArea pantalla;
19 public Bezier(){
20         initComponents();
21 }
22 private void initComponents(){
23         panel=new Panel();
24         panelc=new Panel();
25         panel2=new Panel();
26         canvas=new MyCanvas();
27         btnLimpiar=new Button("Limpiar");
28         pantalla=new TextArea("", 30, 15);
29         calidad=new Checkbox("Curva alta calidad",false);
30         desliz=new Checkbox("Mostrar/Ocultar puntos",true);
31         pol=new Checkbox("Mostrar/Ocultar poligono Bezier",false);
32         btnAnadir=new Button("Añadir (MAX 450,450)");
33         px=new TextField("X",2);
34         py=new TextField("Y",2);panel.setBackground(Color.lightGray);
35         panel2.setBackground(Color.gray);
36         canvas.setBackground(Color.white);
37         canvas.setBounds(0,0,450,450);
38         pantalla.setEditable(false);
39         addWindowListener(new java.awt.event.WindowAdapter(){
40                         public void windowClosing(java.awt.event.WindowEvent evt)
                               {
41                                 exitForm(evt);
42                         }
43         });
44         btnLimpiar.addActionListener(new ActionListener() {
45                         public void actionPerformed(ActionEvent evt) {
46                                 btnLimpiarActionPerformed(evt);
47                         }
48         });
49         btnAnadir.addActionListener(new ActionListener() {
50                         public void actionPerformed(ActionEvent evt) {
51                                 btnAnadirActionPerformed(evt);
52                                 }
53         });
54         calidad.addItemListener(this);
```

```
55          desliz.addItemListener(this);
56          pol.addItemListener(this);
57          add(btnLimpiar, BorderLayout.SOUTH );
58          panel.add(pantalla);
59          panel2.add(px);
60          panel2.add(py);
61          panel2.add(btnAnadir);
62          panel2.add(calidad);
63          panel2.add(desliz);
64          panel2.add(pol);
65          panel.setBounds(0,70,800,600);
66          pantalla.setBounds(0,110,200,200);
67          panelc.add(canvas);
68          panelc.setBounds(200,0,200,500);
69          add(panel);
70          add(panelc);
71          add(panel2);
72          panel.add(panelc);
73          pack();}
74          public void itemStateChanged(ItemEvent e){
75                  checkHandler();
76          }
77          public void checkHandler(){
78                          if(calidad.getState())
79                                  canvas.modificarCalidad(0.001f);
80                          else
81                                  canvas.modificarCalidad(0.01f);
82                          if(desliz.getState())
83                                  canvas.setMostrarAux(true);
84                          else
85                                  canvas.setMostrarAux(false);
86                          if(pol.getState())
87                                  canvas.setMostrarPol(true);
88                          else
89                                  canvas.setMostrarPol(false);
90          }
91          private void btnLimpiarActionPerformed(ActionEvent evt) {
92                  canvas.limpiar();
93          }
94          private void btnAnadirActionPerformed(ActionEvent evt) {
95                  String x=px.getText();
96                  String y=py.getText();
97                  Point p=new Point(Integer.parseInt(x),Integer.parseInt(y));
98                  canvas.anadirPunto(p);
99          }
100         private void exitForm(java.awt.event.WindowEvent evt){
101                 System.exit(0);
102         }
103         static public void modificarTexto(String s){
104                  pantalla.setText(s);
105         }
106
107         public static void main(String[] args){
108                 Bezier marco =new Bezier();
109                 marco.setSize(800,600);
110                 marco.setTitle("Editor Curvas Bezier");
111                 marco.setVisible(true);
112         }
113 }
```

CODIGO PROGRAMA EN JAVA

```
1
2  import java.awt.Canvas;
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import java.awt.Point;
6  import java.awt.event.MouseAdapter;
7  import java.awt.event.MouseEvent;
8  import java.awt.event.MouseMotionAdapter;
9  import java.util.Vector;
10
11 class MyCanvas extends Canvas{
12         private Vector[] puntos,curva;
13         private Point puntoSeleccionado;
14         private int numCurvas;
15         private float calidad;
16         private boolean mostrarAux;
17         private boolean mostrarPol;
18
19         public  MyCanvas(){
20
21                 puntos=new Vector[20];
22                 curva=new Vector[20];
23                 for(int i = 0; i < 20; i++){
24                         puntos[i] =new Vector<Point>();
25                         curva[i] =new Vector<Point>();
26                 }
27
28                 puntoSeleccionado=null;
29                 numCurvas= 0;
30                 calidad= 0.01f;
31                 mostrarAux=true;
32                 mostrarPol=false;
33                 addMouseListener(new MouseHandler());
34                 addMouseMotionListener(new MouseMotionHandler());
35         }
36
37
38 public void paint(Graphics g){
39         for(int pos = 0; pos <=numCurvas; pos++){
40         int n =puntos[pos].size();
41         if(mostrarAux){
42                 for(int i = 0; i < n; i++){
43                         Point p = (Point)puntos[pos].elementAt(i);
44                         g.drawOval((int) p.getX() - 2, (int) p.getY() - 2, 6, 6);
45                 }
46         }
47         int nc =curva[pos].size();
48         for(int i = 0; i < nc; i++){
49                 Point p = (Point)curva[pos].elementAt(i);
50                 g.setColor(Color.red );
51                 g.fillOval((int) p.getX(), (int) p.getY(), 2, 2);
52                 g.setColor(Color.black);
53         }
54         if(n >= 2 &&mostrarAux){
55                 g.drawLine((int) ((Point)puntos[pos].elementAt(0)).getX(),(int)
                       ((Point)puntos[pos].elementAt(0)).getY(),(int) ((Point)puntos[
                       pos].elementAt(1)).getX(),(int) ((Point)puntos[pos].elementAt
                       (1)).getY());
```

```
56          }
57          if(n == 3){
58                  if(mostrarAux){
59                          g.drawLine((int) ((Point)puntos[pos].elementAt(1)).getX()
                                ,(int) ((Point)puntos[pos].elementAt(1)).getY(),(int)
                                ((Point)puntos[pos].elementAt(2)).getX(),(int) ((Point
                                )puntos[pos].elementAt(2)).getY());
60                  }
61                  if(mostrarPol){
62                          g.setColor(Color.blue);
63                          g.drawLine((int) ((Point)puntos[pos].elementAt(0)).getX()
                                ,(int) ((Point)puntos[pos].elementAt(0)).getY(),(int)
                                ((Point)puntos[pos].elementAt(1)).getX(),(int) ((Point
                                )puntos[pos].elementAt(1)).getY());
64                          g.drawLine((int) ((Point)puntos[pos].elementAt(1)).getX()
                                ,(int) ((Point)puntos[pos].elementAt(1)).getY(),(int)
                                ((Point)puntos[pos].elementAt(2)).getX(),(int) ((Point
                                )puntos[pos].elementAt(2)).getY());
65                          g.setColor(Color.black);
66                  }
67          }
68          }
69  }
70
71  public void bezier(){
72          mandarTexto();
73          for(int pos = 0; pos <numCurvas; pos++){
74                  float p = 0.0f, x0 = 0.0f, y0 = 0.0f, x1 = 0.0f, y1 = 0.0f, x2 =
                        0.0f, y2 = 0.0f,x3 = 0.0f, y3 = 0.0f;
75                  float tx = 0.0f, ty = 0.0f;
76                  int n =puntos[pos].size();
77                  if(n <= 1)
78                          return;
79                  Point pto = (Point)puntos[pos].elementAt(0);
80                  x0 += pto.getX();
81                  y0 += pto.getY();
82
83                  pto = (Point)puntos[pos].elementAt(1);
84                  x1 += pto.getX();
85                  y1 += pto.getY();
86
87                  pto = (Point)puntos[pos].elementAt(2);
88                  x2 += pto.getX();
89                  y2 += pto.getY();
90                  while(p <= 1){
91                          tx = exp((1 - p), 2) * x0 + 2 * p * exp((1 - p), 1) * x1
                                 + exp(p, 2) * x2;
92                          ty = exp((1 - p), 2) * y0 + 2 * p * exp((1 - p), 1) * y1
                                  + exp(p, 2) * y2;
93                          curva[pos].addElement(new Point((int) tx, (int) ty));
94                          p +=calidad;
95                  }
96                  repaint();
97          }
98
99  }
100
101
102
103
```

```
104 private float exp( float base,int exp){
105         float r = 1.0f;
106         int i = 0;
107         for(; i < exp; i++){
108                 r *= base;
109         }
110         return r;
111 }
112
113 public void limpiar(){
114         for(int i = 0; i <=numCurvas; i++){
115                 if(puntos[i] !=null)
116                         puntos[i].removeAllElements();
117                 if(curva[i] !=null)
118                         curva[i].removeAllElements();
119                 }
120         numCurvas= 0;
121         repaint();
122 }
123
124 public void limpiarCurva(){
125         for(int i = 0; i <curva.length; i++){
126                 curva[i].removeAllElements();
127         }
128         repaint();
129 }
130
131 public void modificarCalidad(float cal){
132         calidad= cal;
133         limpiarCurva();
134         bezier();
135 }
136
137 public  void mandarTexto(){
138         String textoPantalla ="";
139         for(int i = 0; i <numCurvas; i++){
140                 textoPantalla +="La curva "+ (i + 1) +" :\n";
141                 textoPantalla +="P1: (";
142                 textoPantalla += ((Point)puntos[i].elementAt(0)).getX();
143                 textoPantalla +=",";
144                 textoPantalla += ((Point)puntos[i].elementAt(0)).getY();
145                 textoPantalla +=")\nP2: (";
146                 textoPantalla += ((Point)puntos[i].elementAt(1)).getX();
147                 textoPantalla +=",";
148                 textoPantalla += ((Point)puntos[i].elementAt(1)).getY();
149                 textoPantalla +=")\nP3: (";
150                 textoPantalla += ((Point)puntos[i].elementAt(2)).getX();
151                 textoPantalla +=",";
152                 textoPantalla += ((Point)puntos[i].elementAt(2)).getY();
153                 textoPantalla +=")";
154                 }
155         Bezier.modificarTexto(textoPantalla);
156 }
157
158
159
160
161
162
163
```

```
164 public void anadirPunto(Point p){
165         if(numCurvas< 19){
166                 puntos[numCurvas].addElement(p);
167                 repaint();
168                 if(puntos[numCurvas].size() == 3){
169                         numCurvas++;
170                         bezier();
171                         puntos[numCurvas].addElement(puntos[numCurvas- 1].
                                elementAt(2));
172                 }
173         }
174 }
175 
176 public void setMostrarAux(boolean b){
177         mostrarAux= b;
178         bezier();
179         }
180 
181 public void setMostrarPol(boolean b){
182         mostrarPol= b;
183         bezier();
184 }
185 
186 
187 class MouseHandler extends MouseAdapter{
188 
189 public void mousePressed(MouseEvent e){
190         puntoSeleccionado= dentroPunto(e.getX(), e.getY());
191 }
192 
193 public void mouseReleased(MouseEvent e){
194         puntoSeleccionado=null;
195 }
196 
197 public void mouseClicked(MouseEvent e){
198         if (numCurvas< 19){
199                 puntos[numCurvas].addElement(new Point(e.getX(), e.getY()));
200                 repaint();
201                 if(puntos[numCurvas].size() == 3){
202                         numCurvas++;bezier();
203                         puntos[numCurvas].addElement(puntos[numCurvas- 1].
                                elementAt(2));
204                 }
205         }
206 }
207 
208 }
209 class MouseMotionHandler extends MouseMotionAdapter{
210 public void mouseDragged(MouseEvent _event){// Si hay algun punto seleccionado lo
        arrastro modificandole // las coordenadas igual a las del raton y repintando
      la pantalla
211         if(puntoSeleccionado!=null){
212                 puntoSeleccionado.setLocation(_event.getX(), _event.getY());
213                 repaint();
214                 limpiarCurva();
215                 bezier();
216         }
217 }
218 
219 }
```

```
220 private Point dentroPunto(int x,int y){
221         for(int pos = 0; pos <numCurvas; pos++){
222                 int n =puntos[pos].size();
223                 int _x = 0, _y = 0;
224                 for(int i = 0; i < n; i++){
225                         Point pto = (Point)puntos[pos].elementAt(i);_x = (int)
                            pto.getX();_y = (int) pto.getY();
226                         if((x > (_x - 3)) && (x < _x + 3) && (y > (_y - 3))&& (y
                            < _y + 3))
227                                 return pto;
228                         }
229                 }
230 return null;
231 }
232 }
```

Aquí tendríamos un ejemplo de la visualizacion del polinomio. Aquí

Para los diferentes puntos hemos obtenido diferentes curvas de Bézier que hemos transformado para poder ponerlo en práctica a la hora de la visualización del juego.

3.4. Visualización del Sprite

En esta seccion vamos a ver la visualización de los sprites del boomerang representados por diferentes métodos de aproximación o interpolación.

En el caso de los polinomios de Lagrange:

- POL LAGRANGE

Polinomios	Imagen
Polinomio 1	
Polinomio 2	
Polinomio 3	
Polinomio 4	

En el caso de las curvas de Bézier:

- CURVAS DE BÉZIER

Curva	Imagen
Curva 1	
Curva 2	
Curva 3	
Curva 4	

Capítulo 4

Conclusiones

Para finalizar, en este último apartado vamos a destacar las diferencias entre los polinomios de Lagrange y los polinomios de Bernstein, tanto en coste computacional como en utilidad.

Los polinomios de Lagrange son muy fáciles de calcular. Es por ello que se utilizan como uno de los primeros ejemplos de polinomios interpoladores. Su interés práctico es limitado y suelen presentarse más bien como ejemplo teórico de interpolación. Su principal inconveniente se presenta cuando el conjunto de nodos es muy grande. En ese caso el grado del polinomio también es muy grande. Esto implica dificultades para el cálculo y, además, hay una alta tendencia a que el polinomio oscile mucho entre dos nodos.

Por otra parte, los polinomios de Bernstein son más complejos que los polinomios de Lagrange pero facilitan el cálculo de polinomios en los que intervienen una gran cantidad de puntos.

Coste Computacional

Tomando como referencia el código de los programas implementados, podemos observar que la creación de un polinomio de Lagrange es más costosa computacionalmente que un polinomio de Bernstein. Esto es debido a que en el codigo sobre los polinomios de Lagrange aparece un bucle anidado (consiste en meter un bucle dentro de otro) mientras que en el otro solo hay un solo bucle "for". Por tanto el coste computacional es de orden $O(n^2)$ para Lagrange y es de orden $O(n)$ para Bernstein.

Bibliografía

Daniel, M Duabisse, J.C. The numerical problem of using Bézier curves and surfaces in the power basis. Computer Aided Geometric Design 6 (1989), pp. 121-128.

Farouki, R.T. Rajan, V.T On the numerical condition of polynomials in Bernstein form.Computer Aided Geometric Design 4 (1987), pp. 191-216.

Farin G.: Curves and Surfaces for computer aided geometric design. A practical Guide. Academic Press 1988, pp. 14-46.

Anand V.B.: Computer Graphics and Geometric Modelling for Engineers. John Wiley Sons, 1993.

3-D Computer Graphics. A Mathematical Introduction with OpenGL, de S. R. Buss

Böhm84 Böhm W.; Farin G.; Kahmann J.: A survey of curve and surface methods in CAGD. Computer Aided Geometric Design. Vol.1, N.1, 1984. pp. 1-60

Barsky B.A.: Computer Graphics and Geometric Modelling Using Beta-splines. SpringerVerlag 1988