**Fase 1: Identificación el Problema y Requerimientos.**

Juego que permita entretener al usuario. El juego se centra el elegir un caballo entre cinco. Apostar una cantidad al caballo y observar si el caballo elegido es el ganador. Solo se le paga al caballo que gane y al segundo.

**Necesidades del usuario:**

1. **Importancia del Ocio:** Las personas necesitan una manera de distraerse. Permitiéndoles disfrutar de un breve tiempo libre jugando a un juego.
2. **Facilidad de Uso y Entretenimiento:** Los usuarios necesitan una interfaz gráfica que les permitan interactuar con el juego y observar el transcurso del juego.
3. **Diversidad del Juego**: Se necesita que el juego tenga varios caballos a elegir y que cada caballo pueda ganar. De esta manera, es más entretenido.
4. **Variedad de apuestas:** Es necesario que el usuario pueda elegir la cantidad a apostar por el caballo que eligió.
5. **Recarga de Dinero:** El usuario debe de tener la opción de “recargar” dinero para las apuestas.
6. **Visualización de Resultados:** El usuario debe poder ver claramente que caballo gano, si gano o perdió y cuánto dinero gano o perdió.

**Requerimientos Funcionales:**

1. **Elegir Caballo:**
   * El sistema debe permitir al usuario elegir entre cinco caballos. El usuario debe elegir el caballo antes de que inicie la carrera.
   * El sistema debe verificar que el usuario tenga fondos para la apuesta mínima.
   * Después de elegir el caballo, el sistema debe permitir al usuario elegir su apuesta.
2. **Elegir Apuesta:**
   * El sistema debe permitir al usuario elegir su apuesta. Esta apuesta debe cumplir los siguientes requisitos:
     + Tiene que ser mayor a 50 dólares y debe ser menor o igual a 10.000 dólares.
     + Tiene que ser en múltiplos de 50 dólares, es decir; no puede apostar 70 dólares, por ejemplo.
3. **Cambiar Caballo o Apuesta:**
   * Antes del inicio de la carrera, se debe permitir al usuario cambiar el caballo, la apuesta o ambos.
4. **Carrera:**
   * Antes de iniciar la carrera, se informa al usuario del caballo elegido y la apuesta realizada, pidiendo una confirmación para iniciar la carrera. Si el usuario confirma, la carrera inicia, por otro lado, si no confirma, permite al usuario cambiar el caballo o la apuesta.
   * Cuando la carrera termine, se debe informar al usuario que caballo gano, y cuánto dinero gano o perdió.
5. **Pago Carrera:**
   * Cuando se acabe la carrera, se revisa la posición del caballo por el cual se aposto:
     + Si el caballo elegido queda de primero, este caballo gana. Se paga lo apostado por 4.7. Es decir, si se aposto la mínima (50 dólares), se hace el siguiente calculo: . El jugador gana en total 235 dólares.
     + Si el caballo elegido queda en segunda posición, colocado. Se paga lo apostado por . Es decir, si se aposta la mínima (50 dólares), se hace el cálculo del ganador, dando como resultado 235 dólares. A este resultado, se le resta la apuesta, . Este valor, se divide entre tres y se le suma la apuesta realizada: . Se realiza la función de suelo, es decir, se quita el decimal y se aproxima el valor al múltiplo de 10 más cercano. . El jugador gana en total 110 dólares.
     + Si el caballo elegido no queda de primero ni de segundo. El jugador pierde el dinero apostado.
6. **Recargar Dinero:**
   * El sistema debe permitir al usuario realizar una recarga de dinero. Esta recarga debe:
     + Ser mínimo de 250 dólares y de un máximo de 100.000 dólares.
     + Ser de múltiplos de 10 dólares, es decir, no puede recargar 122 dólares, por ejemplo.
   * Para recargar dinero, el usuario debe no estar participando en una carrera.
7. **Visualizar Resultados Anteriores:**
   * El sistema debe informar al usuario del resultado de todas las carreras en las que ha participado. Se informa a través de la cantidad de carreras que ha ganado cada caballo y a través del porcentaje de victoria que tiene cada caballo. Es decir, caballo “x” ha ganado 4 carreras, ha ganado el 20% de todas las carreras.
8. **Permitir Cambiar Algoritmo:**
   * El sistema debe permitir al usuario cambiar el algoritmo que se va a utilizar en la carrera. Cambiar el algoritmo de Dijkstra al algoritmo de Floyd-Warshall y viceversa.

**Fase 2: Investigación.**

**Definiciones:**

1. **¿Cómo se gana?:**

Se gana eligiendo el caballo que quede en primera posición o se gana un dividendo si el caballo elegido queda de segundo. La puesta sobre el caballo ganador se paga por 4.7. La apuesta por el segundo es la cuota colocada de .

1. **¿Como se calcula que caballo va a ganar?:**

Este juego cuenta con un algoritmo aleatorio realizado con base en teoría de grafos. De esta manera el caballo que gana es completamente aleatorio. Sin embargo, se muestran los últimos ganadores, por si el usuario se quiere basar en las probabilidades que ha tenido cada caballo en ganar.

1. **Grafo:**

Un grafo es un conjunto de objetos que tienen pueden tener relación entre ellos. Un grafo está formado por vértices y por aristas que conectas los vértices entre ellos. Estas aristas pueden ser dirigidas o no dirigidas. A la vez, cada arista puede tener un peso o no tener peso.

Los grafos dirigidos se caracterizan por tener aristas asociadas a una dirección (las aristas no son equivalentes). Por otro lado, los grafos no dirigidos tienen la característica que las aristas son equivalentes .

En este caso, se va a utilizar un grafo que la principal característica es que las aristas tienen un peso.

1. **Pista como grafo:**

Algunos juegos de carreras tienen un sistema de “checkpoint”. Esto significa que la pista se divide en distintos puntos de control. Dando como resultado que el jugador debe pasar por todos los puntos de control antes de pasar por la meta para que la vuelta cuente. Un ejemplo para entender el concepto es el circuito hamiltoniano; es un circuito que visita cada vértice una vez sin repeticiones, este inicia y termina en el mismo vértice.

Con estos conceptos se generó la idea de crear una pista que inicie en un vértice, y pueda pasar por distintos vértices, que tienen peso, que conforman la pista, llegando a un vértice final.

1. **Algoritmos de Dijkstra:**

El Algoritmo de Dijkstra comienza desde el nodo de origen y explora el grafo en busca del camino más corto hacia todos los demás nodos.

El algoritmo realiza un seguimiento de la distancia más corta conocida hasta el momento entre el nodo de origen y cada otro nodo, actualizando estos valores si se descubre un camino más corto.

Una vez que el algoritmo ha identificado el camino más corto entre el nodo de origen y otro nodo, ese nodo se etiqueta como "visitado" y se incorpora al camino.

Este proceso continúa hasta que se hayan visitado todos los nodos del grafo. Como resultado, obtenemos un camino que conecta el nodo de origen con todos los demás nodos, siguiendo la ruta más corta disponible para alcanzar cada nodo.

1. **Algoritmo de Floyd-Warshall:**

El algoritmo de Floyd-Warshall, se centra en encontrar el camino más corto entre todos los vértices en un grafo con peso. Este algoritmo funciona para grafos dirigidos y no dirigidos.

Funciona mediante la construcción de una matriz que contiene las distancias más cortas entre los nodos. Luego, compara todas las rutas posibles a través de todos los nodos para actualizar la matriz con los caminos más cortos. Esto se repite hasta que se obtenga la matriz final de distancias más cortas. El algoritmo es eficiente para grafos pequeños y funciona incluso cuando hay ciclos negativos en el grafo.

**Fase 3: Búsqueda de ideas creativas.**

**Lluvia de ideas:**

1. **Grafos como pista:** Crear una pista que inicie en un vértice, y pueda pasar por distintos vértices que estén conectados por aristas y que cada arista tenga un peso. Dando como resultado, que la pista inicie en un vértice y con distintos algoritmos se encuentre el camino hasta el último vértice.
2. **Mapa de detección de colisiones:** Se crea la pista mediante el uso de un software de procesamiento de imágenes. Luego se cambia la pista a un color para establecer la detección de colisiones. Todo lo demás en el mapa se pinta de blanco. De esta manera la pista es representada con estos colores generando una pista con sus colisiones.
3. **Escaneo LIDAR:** Algunas pistas se suelen crear con datos LIDAR (Light Detection and Ranging) para mapear entornos del mundo real y crear pistas basadas en ubicaciones reales.
4. **Diseño manual y editor de niveles:** Los diseñadores de pistas pueden crear pistas manualmente, utilizando un software de diseño de niveles. Esto implica dibujar y colocar elementos en la pista de forma manual.
5. **Escaneo fotográfico:** Algunos juegos suelen crear pistas utilizando técnicas de escaneo fotográfico. De manera que se pueden capturar detalles del mundo real y aplicarlos en pistas en proceso de creación.
6. **Generación procedural:** En algunos juegos, las pistas se generan de forma procedural utilizando algoritmos en específico. Esta técnica permite crear pistas únicas cada vez que se juega. Se utilizan técnicas y algoritmos como el ruido Perlin, fractales o automatons celulares.

**Fase 4: Transición de la formulación de ideas de diseños preliminares.**

Se descartan las ideas del escaneo LIDAR, funciona para mapear entornos del mundo real, cosa que es innecesaria y no es posible realizarla. El escaneo fotográfico se descarta por al no ser necesario tener una pista con detalles del mundo real. Y, por último, la generación procedural, al no ser necesario una pista nueva cada que se juegue una partida.

**Ideas:**

* **Alternativa 1. Grafos como pista:**

Esta alternativa es la manera en la cual funcionara la estructura no visible de la pista. Es la manera necesaria para que el juego cumpla con los requisitos que se necesitan para el funcionamiento interno del juego. Se conocen varios juegos que hacen uso de los grafos para creación de pista.

* **Alternativa 2. Mapa de detección de colisiones:**

El punto fuerte de esta alternativa es la capacidad de identificar colisiones en la pista. Es decir, la jugabilidad del videojuego se ve mejorada por el funcionamiento del mapa de detección de colisiones que ofrece esta alternativa. Esta alternativa funcionaria con los colores que se explico anteriormente. Siendo desarrollada con una matriz.

* **Alternativa 4. Diseño manual y editor de niveles:**

Esta alternativa es una idea que va a ser tomada en cuenta. Permite, en el desarrollo de juegos sin o con poco presupuesto, un desarrollo de las pistas que van a ser utilizadas en el juego. Este desarrollo se centra en dibujar y colocar elementos en la pista de forma manual.

**Paso 5. Evaluación y Selección de la Mejor Solución.**

Criterios:

1. Facilidad en implementación:

Fácil de implementar: 1

Complejidad regular: 2

Difícil de implementar: 3

1. Tiempo que consume:

Poco tiempo: 3

Tiempo moderado: 2

Gran cantidad de tiempo: 1

1. Personalización:

Muy personalizable: 1

Poco personalizable: 2

Nada personalizable: 3

1. Presupuesto necesario:

Poco costosa: 1

Normal: 2

Costosa: 3

1. Funcionalidad:

Altamente funcional para el problema: 1

Funcionalidad moderada para el problema: 2

Poco funcional para el problema: 3

|  |  |  |  |
| --- | --- | --- | --- |
| Solución | Grafos como pista | Mapa de detección de colisiones | Diseño manual y editor de niveles |
| Criterios |
| Facilidad en implementación | 1 | 3 | 2 |
| Tiempo que consume | 1 | 3 | 2 |
| Personalización | 2 | 1 | 1 |
| Presupuesto necesario | 1 | 2 | 2 |
| Funcionalidad | 1 | 3 | 1 |
| Total | 6 | 12 | 8 |

Como los criterios son 1 para el mejor y 3 para el peor, entonces el total más bajo es el mejor. Esto nos arroja que los grafos como pista es la mejor solución.