**Kitten Survival**

**Juan Felipe Vásquez Bolívar & Jackh Emmanuel Narváez Guerra**

**C.C 1001509367 & C.C 1002752230**

**Augusto Enrique Salazar Jiménez**

**mayo 2023**

**Universidad de Antioquia**

**Facultad de Ingeniería**

**Informática II**

# **Introducción**

El curso de informática 2 nos ha llevado a exigirnos en muchos sentidos, llevando a cabo diferentes desafíos. De esta manera, hemos ido desarrollando habilidades a nivel personal y profesional. Se nos ha puesto el reto de dejar volar la imaginación para idear un juego que después tendremos que materializar utilizando el lenguaje de programación C++ y herramientas de Qt. El entregable es para describir la idea que tenemos para realizar el proyecto final y posteriormente ir realizando correcciones.

Las ideas han ido surgiendo a medida que se ha ido desarrollando este trabajo. Hemos dejado que nuestra imaginación hable al momento de crear ideas, personajes, entornos, enemigos, entre otros.

El documento contiene la descripción del tipo de juego a desarrollar, la historia que hemos inventado, la manera en que le daremos flujo al juego y cuáles son los objetivos que queremos lograr. Por último, se incluyen las conclusiones que obtuvimos después de realizar este entregable.

El reto que estamos afrontando será un "borrador" para llevar a cabo nuestro proyecto final, el cual, con nuestras capacidades, lograremos completar exitosamente.

# **Descripción**

Eres un gato hambriento que necesita comida desesperadamente. En Kitten Survival, tu objetivo es infiltrarte en una casa y encontrar comida sin ser detectado por los enemigos. Una vez que hayas encontrado la comida, tendrás que escapar de la casa sin ser atrapado.

**Modo de juego:** Kitten Survival es un juego bidimensional con una perspectiva desde arriba. Tendrás que moverte horizontal y verticalmente por la casa para encontrar la comida que necesitas. El juego se divide en dos etapas: la infiltración y el escape.

En la etapa de infiltración, tendrás que encontrar la comida sin ser detectado por los enemigos. Puedes esconderte detrás de muebles, moverte silenciosamente y evitar la línea de visión de los enemigos. Si te detectan, tendrás que correr y esconderte hasta que los enemigos se den por vencidos.

En la etapa de escape, tendrás que escapar de la casa con la comida que has encontrado. Tendrás que evitar a los enemigos y cualquier obstáculo que se interponga en tu camino. Si te atrapan, perderás la comida y tendrás que empezar de nuevo.

A medida que avances en el juego, encontrarás diferentes tipos de comodines que te otorgaran habilidades especiales que te ayudaran con la infiltración y escape. También habrá enemigos más difíciles de evitar, por lo que tendrás que ser más astuto y rápido para sobrevivir.

**Menú**

En el menú encontraremos diversas opciones el cual te facilitara disfrutar del juego, algunas opciones son:

**1. Nuevo juego:** Esta será la primera opción que visualizará el usuario, esta los llevará a iniciar con el primer nivel del juego cada vez que un nuevo usuario desee jugar, también quienes ya jugaron el juego tendrán esta opción por si desean comenzar de nuevo.

**2. Continuar con el juego:** Esta será la segunda opción que podrá observar el usuario que ya haya jugado y desee continuar con la partida, es decir que quienes ingresan por primera vez no podrán observar esta opción.

**3. Salir:** Esta será la última opción que aparecerá en el menú, llevar a que el usuario se pueda salir del juego sin ningún problema.

**Movimientos**

Uno de los desafíos es llevar la física que observamos día a día en nuestro alrededor al juego en forma de simulación donde utilizaremos conceptos como la cinemática o la fluidez y algunos movimientos de traslación. A continuación, nombres de algunos movimientos y para que los pudiéramos utilizar

**1. Fricción:** Pretendemos usar la fricción en la implementación de nuestro juego, creando zonas que dificultan la movilidad de nuestro personaje.

F\_fricción\_estática = μ\_s \* N, Donde F\_fricción\_estática es la fuerza de fricción estática, μ\_s es el coeficiente de fricción estática entre los objetos y N es la fuerza normal que actúa perpendicular a la superficie.

**2. Movimiento circular uniforme (M.C.U):** Durante el mapa encontraremos comodines cuyo desplazamiento representará un M.C.U, que dificultará su obtención. Estos comodines otorgaran beneficios a nuestro personaje principal.

La fórmula matemática para la posición del objeto en coordenadas polares es la siguiente:

**r = constante**

**θ = ω \* t**

donde r es la distancia desde el origen al objeto, θ es el ángulo que el objeto forma con el eje horizontal, ω es la velocidad angular del objeto, t es el tiempo y la constante es la distancia radial desde el origen al objeto.

Para convertir estas coordenadas polares en coordenadas cartesianas, se puede utilizar la función coseno y seno. La fórmula para la posición en coordenadas cartesianas es la siguiente:

**x = r \* cos(θ)**

**y = r \* sin(θ)**

**3. Movimiento Rectilíneo Uniforme (M.R.U):** Nuestro personaje principal y algunos enemigos se moverán a una velocidad constante, describiendo así un M.R.U.

**4. Movimiento Rectilíneo Uniformemente Acelerado (M.R.U.A):** Con el objetivo de agregar dificultad al juego, algunos enemigos contarán con una aceleración, la cual causará que a medida que pase el tiempo, su movimiento será mas veloz.

**Clases Para Implementar**

**Clase Principal:**

**1. Clase Sprite:** Esta clase heredará la clase **<QObject>** y <**QGraphicsPixmapItem**> será la encargada de dar manejo a todos los Sprites implementados en el videojuego, ya que permite recortar y seleccionar un Sprite.

**1.1 Atributos:**

**QPixmap \***pixmap**, \***current**\_**pixmap**;**

**1.2 Métodos:**

**void set\_imagen(int** ancho**, int** alto**, int** x\_origin**, int** y\_origin**, int** x\_sprites**, int** y\_sprites**, QString** ubicacion**); //Permite recortar un sprite para obtener un QPixmap que obtenga solo los Sprite que necesita.**

**void select\_sprite(int** x**, int** y**); //Del nuevo sprite obtenido, permite seleccionar el cual queremos mostrar en pantalla.**

**Clases Secundarias:**

**1.Clase mapa:** recibe el ancho y alto de la clase mapa y crea una matriz bidimensional para almacenar los caracteres que representan el mapa del juego y además heredará la clase propia “Sprite.h”.

**1.1 Atributos:**

Mapa(int ancho, int alto);

void Dibujar();

void ModificarCasilla(int x, int y, char valor);

**1.2 Métodos:**

Void Bloques**(int** ancho**, int** alto**, int** x\_origin**, int** y\_origin**, int** x\_sprites**, int** y\_sprites**, QString** ubicacion**);**

**2.Clase Bloques:**El constructor de la clase se encarga de inicializar los atributos a partir de los valores proporcionados al crear un objeto Bloque. y además heredará la clase propia “Sprite.h”.

**Atributos:**

x\_, y\_, ancho\_ y alto\_, **//representan la posición y las dimensiones del bloque**

**Métodos:**

**Void bloque** ((ObtenerX(), ObtenerY(), ObtenerAncho(), ObtenerAlto()); **//permiten acceder a los atributos privados desde fuera de la clase.**

**3.Clase de jugador:** Esta clase es esencial, ya que representa al personaje principal o al jugador en el juego, por lo tanto, heredará la clase propia “Sprite.h”.

**3.1 Atributos:**

Esta clase no cuenta con atributos.

**3.2 Métodos:**

**void tecla\_pressed(QKeyEvent \***event**); // Esta función es la encargada del movimiento del personaje principal**

**4. Clase de enemigo\_1:** Esta clase representa a un tipo de personaje que actúa como adversario del jugador. Esta clase representa los enemigos que tienen un movimiento rectilíneo uniforme y además heredará la clase propia “Sprite.h”.

**4.1 Atributos:**

Esta clase no cuenta con atributos.

**4.2 Métodos:**

**void Movimiento\_Rectilineo(); // Esta función es la encargada del movimiento de esta clase de enemigo.**

**5. Clase de enemigo\_2:** Esta clase representa a un tipo de personaje que actúa como adversario del jugador. Esta clase representa los enemigos que tienen un movimiento rectilíneo uniformemente acelerado y además heredará la clase propia “Sprite.h”.

**5.1 Atributos:**

Esta clase no cuenta con atributos.

**5.2 Métodos:**

**void Movimiento\_Acelerado(); // Esta función es la encargada del movimiento de esta clase de enemigo.**

**6. Clase de Comodín:** Esta clase representa a un tipo de personaje que actúa como comodín que otorga beneficios a nuestro personaje principal. Esta clase representa los enemigos que tienen un movimiento circular uniformemente y además heredará la clase propia “Sprite.h”.

**6.1 Atributos:**

Esta clase no cuenta con atributos.

**6.2 Métodos:**

**void Movimiento\_Circular(); // Esta función es la encargada del movimiento de este comodin.**

**Clase de nivel:** Esta clase se utiliza para crear y manejar los diferentes niveles o etapas del juego. Los niveles pueden incluir diferentes elementos, como terrenos, obstáculos, enemigos y recompensas. Además, heredará la clase propia “Sprite.h”.

**Clase Definitiva:**

**1. Clase Juego:** Esta clase importará todas las clases propias del juego, y será el encargado de realizar la lógica del mismo, por lo tanto, será quien determine cuando dos objetos colisionan, las consecuencias que tendrá, entre otros.

**1.1 Atributos:**

**Jugador \***P\_Ppal**;**

**Enemigo\_1 \***Enemigo1**;**

**Enemigo\_2 \***Enemigo2**;**

**Comodin \***Bonus;

**Nivel \***Mapa;

**QTimer** \*Time;

**1.2 Métodos:**

**Void Mover\_Personaje(QKeyEvent \***event**) // Esta función se encarga de revisar constantemente la posición del personaje principal, e identificar como debe interactuar con el mapa.**

**Void Mover\_Enemigo\_1() // Esta función se encarga de revisar constantemente la posición del enemigo tipo 1, e identificar como debe interactuar con el mapa.**

**Void Mover\_Enemigo\_2() // Esta función se encarga de revisar constantemente la posición del personaje principal, e identificar como debe interactuar con el mapa.**

**Void Mover\_Comodin() // Esta función se encarga de revisar constantemente la posición del comodín principal, e identificar como debe interactuar con el mapa.**