Pengo™

Jorge Lasaosa (542183)
Roberto Clemente (599943)

1. Concepto de juego

El protagonista es un pingüino, PENGO y tiene por enemigos unos seres llamados SNOBEES que tratarán de alcanzarlo. PENGO puede deshacerse de ellos de diversas formas, y cuando no quedan más enemigos en la pantalla se avanza al siguiente nivel. El objetivo del juego es ir avanzando de niveles mientras se intenta lograr la mayor puntuación posible.

Al empezar la partida, aparece una cuadrícula de 13 de ancho x 15 de alto de bloques de hielo contenidos entre cuatro paredes. Acompañado de una música, se van eliminando bloques para formar un laberinto de bloques:

- Bloques de hielo que se pueden empujar y romper
- 3 bloques de diamante que pueden ser alineados vertical u horizontalmente para conseguir puntos extra.
- Bloques de hielo que parpadean al empezar indicando que contienen huevos de SNO-BEE.

Tras comenzar la partida PENGO aparece en el centro de la pantalla y se abren 3 de los huevos de SNOBEE de forma aleatoria.

PENGO se puede mover por todo el laberinto y puede variar la posición de los bloques de hielo que lo forman empujándolos:

- Si PENGO empuja un bloque de hielo “libre” o de diamante (que no está bloqueado en el lado opuesto), el bloque se deslizará hasta chocar con otro bloque o con una de las cuatro paredes.

- Si PENGO empuja un bloque “no libre” (que está bloqueado en el lado opuesto por una pared o por otro bloque), el bloque de hielo se desintegrará (los bloques de diamante no pueden ser desintegrados).

- Si PENGO empuja una pared, ésta vibrará aturdiendo a los SNO-BEES que la toquen.

Las formas de eliminar a los SNO-BEES son dos:

1. Que sea alcanzado por un bloque en movimiento.
2. Que sea alcanzado por PENGO mientras está aturdido.

Al eliminar un SNO-BEE, otro saldrá de alguno de los huevos escondidos en los bloques de hielo. Al acabar con todos los SNO-BEES el nivel es completado y PENGO avanzará al siguiente nivel con una disposición de bloques y SNO-BEES distinta.

Alinear los tres bloques de diamante vertical u horizontalmente es la mejor manera de lograr una puntuación alta por la gran bonificación de puntos que otorga. Además cuando esto sucede todos los SNO-BEES en el campo son aturdidos.

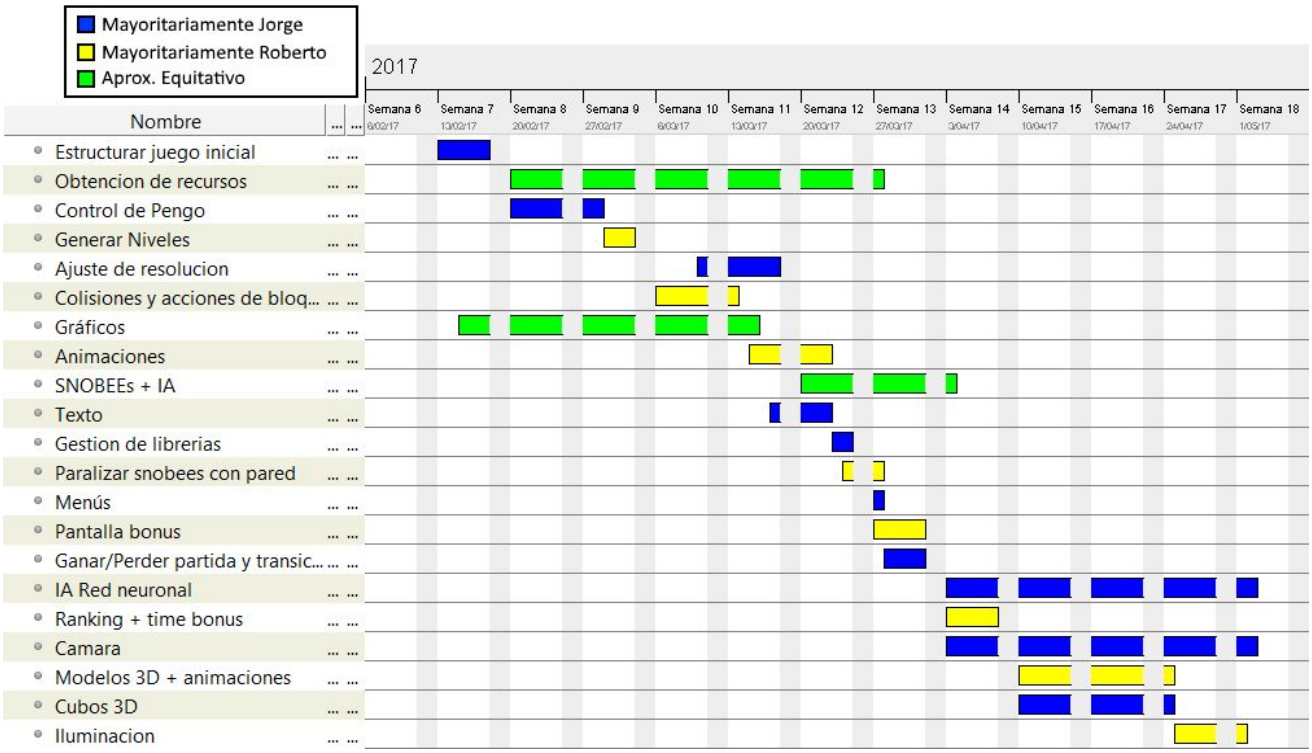
Cuando un SNO-BEE alcanza a PENGO, éste pierde una vida de las tres iniciales. De esta manera, consumiendo una vida, se reanuda el juego manteniendo la disposición de los bloques, pero PENGO reaparece en el centro de la pantalla y los SNO-BEES en una esquina aleatoria. PENGO obtiene una vida extra al alcanzar los 30.000 puntos. El juego termina cuando PENGO pierde todas las vidas.

2. Reparto de roles, tareas y cronograma

Al comienzo del proyecto se repartieron los siguientes roles:

- Responsable de IA: Jorge Lasaosa, pues realiza el TFG sobre Inteligencia Artificial.
- Responsable de gráficos: Roberto Clemente, ya que posee conocimientos sobre edición de imágenes y modelado 3D.
- Responsable de test: Jorge Lasaosa, ya que ha jugado más al videojuego original y tiene mejores referencias.
- Responsable de calidad: Roberto Clemente, pues tiene experiencia en la creación de videojuegos.

Salvo al principio del proyecto, que mientras Roberto se dedicaba a aprender el funcionamiento de openGL y Jorge a establecer una base para el juego, el resto de reparto de tareas se realizó a través de la herramienta Trello. Allí se anotaban las tareas a realizar y cada miembro se asignaba cada tarea a sí mismo con el fin de evitar solapamientos. A pesar de que este método aplicado a tareas cortas proporciona una realización del trabajo bastante homogénea, se pueden clasificar los mayores esfuerzos de cada integrante de acuerdo al siguiente cronograma:

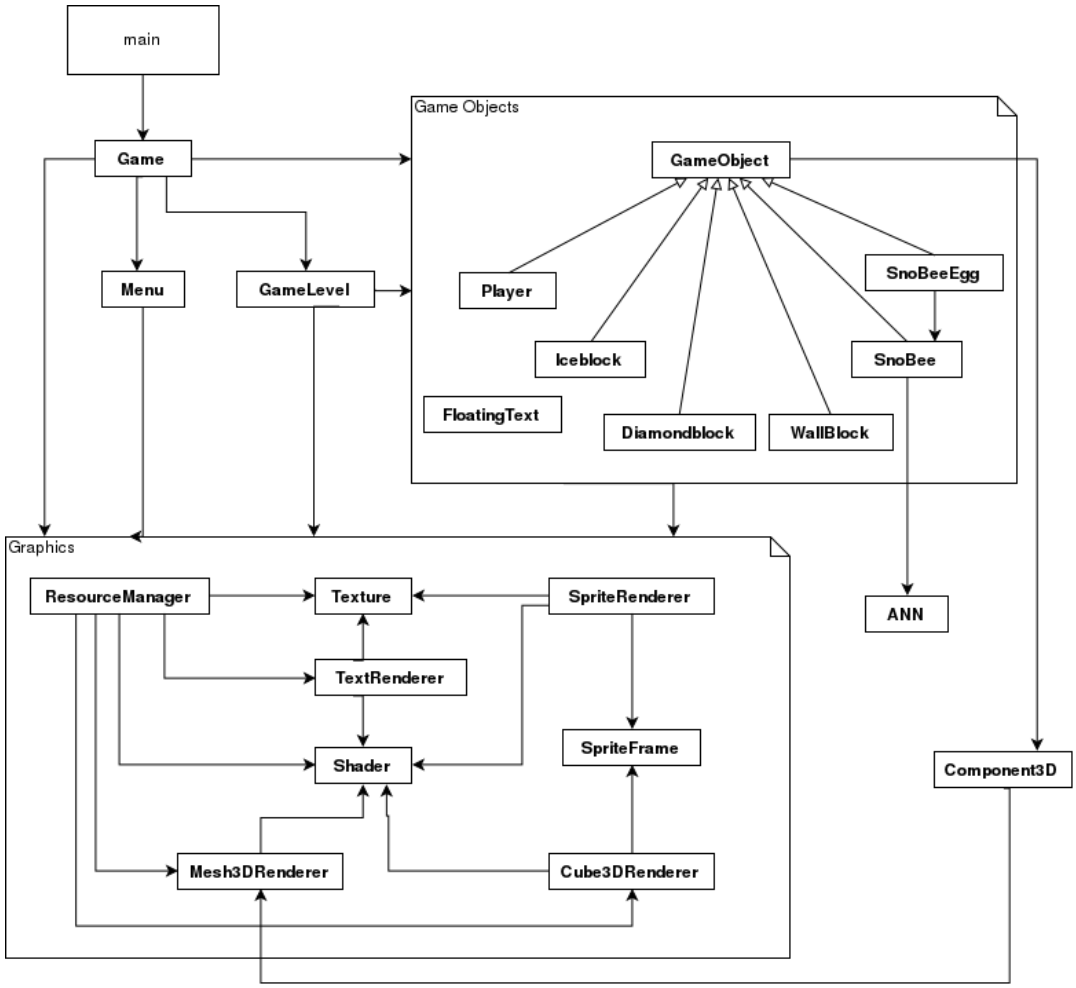


3. Cuestiones iniciales de implementación

Se decidió realizar el juego para la plataforma PC bajo un sistema operativo Linux, pero posteriormente se decidió mantener una portabilidad al sistema operativo Windows. Se decidió programar en C++, ya que ambos integrantes teníamos experiencia en ese lenguaje, es ampliamente utilizado en el mundo de los videojuegos, es versátil y ofrece posibilidad utilizar openGL directamente.

4. Estructura software

Los recursos se encuentran en las carpetas img (imágenes), sounds (sonidos) y levels (niveles). Los shaders utilizados por openGL figuran en la carpeta homónima. Las librerías utilizadas se encuentran dentro del directorio include para acceder a ellas de forma estática. De esta forma, el usuario podrá ejecutar el juego sin necesidad de descargarse las librerías manualmente. El código se encuentra dividido en las carpetas include (ficheros header) y src (contenido de las clases y el main). A continuación se muestra el diagrama de clases del juego:



Los componentes del programa se pueden subdividir en los siguientes grupos (aunque no se han dividido explícitamente en la estructura del programa):

1. Clases principales y de control. Mantienen el flujo principal del juego y su arquitectura base. A él pertenecen las clases:
    a. main: bucle principal e inicializaciones.
    b. Game: componentes globales del juego, configuraciones básicas, entrada de teclado…
    c. GameLevel: información del nivel durante el juego, lógica general durante la partida…
    d. Menu: menú inicial y pause del juego. Acceso a opciones configurables

2. Clases de gráficos. Abstraen de la configuración a bajo nivel de los graficos. Pertenecen las siguientes clases:
    a. ResourceManager: se encarga de gestionar todos los accesos a recursos (texturas, shaders, sonidos...).
    b.
    c. Texture: Guarda la referencia a una textura o sprite.
    d. TextRenderer: Permite escribir texto por pantalla
    e. Shader: Permite utilizar y configurar los shaders.
    f. SpriteRenderer: Permite dibujar una imagen o frame en la pantalla.
    g. SpriteFrame: Permite configurar un frame dentro de un spritesheet.
    h. Mesh3DRenderer: Permite importar y dibujar un modelo 3D.
    i. Cube3DRenderer. Permite dibujar un cubo en 3D con texturas.
3. Clases de objetos del juego. Implementan todas las funcionalidades de los objetos que aparecen durante el juego.
4. Otros:
    a. ANN (Artificial Neural Network). implementa la Inteligencia Artificial de los SNOBEES.
    b. Component3D. Representa cada parte móvil de un objeto en 3D.

5. Gráficos

Para incluir los gráficos del juego se decidió utilizar openGL, ya que es opensource y uno de los integrantes ya sabía utilizarlo. Además se utilizan las librerías GLFW para la creación y uso de ventanas, GLEW para facilitar la ubicación de algunas funciones en tiempo real, GLM para facilitar el uso de vectores y matrices y SOIL para la carga de texturas.

Las texturas de la versión 2D están almacenadas como spritesheets, donde figuran los distintos frames de los elementos gráficos ordenados, a los que se podrá acceder por medio de la clase SpriteFrame. De esta forma resulta sencillo construir animaciones arbitrarias desde dentro del juego, únicamente indicando la posición y tamaño del frame que se desea mostrar. Este método se ha utilizado tanto para las animaciones de los objetos de los niveles (obtenidas de internet) como para la intro y el título del juego (obtenidas mediante capturas directas de un emulador).

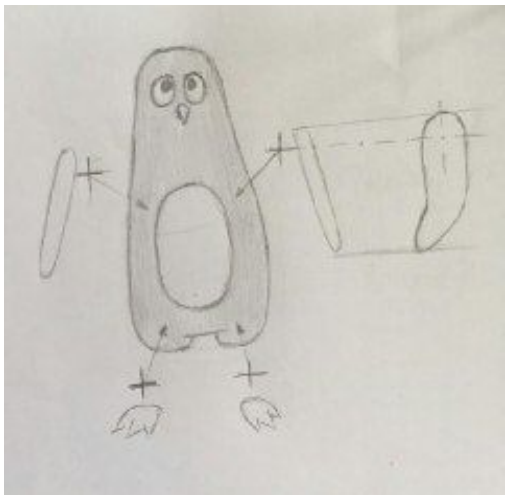
Para pintar texto en pantalla hemos usado la librería FreeType que permite cargar TrueTypeFonts (ttf) en Bitmaps para poder pintar los caracteres como texturas. Igual que hacemos con los sprites creando quads a los que les ponemos una textura.

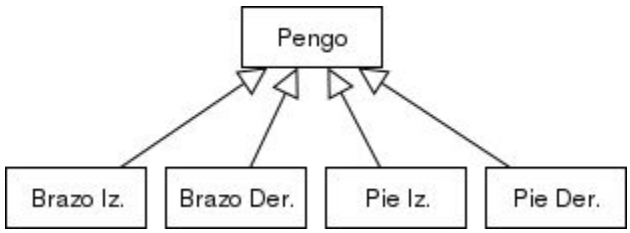
En la versión 3D se tomaron las siguientes decisiones para el uso de modelos 3D más complejos:

- Crear nosotros mismos los modelos de blender para tener mayor control sobre ellos.
- Utilizar el color de los vértices en lugar de texturas para evitar el mapeo de UVs.
- Exportar en formato PLY, pues es el único de los disponibles que permite exportar los colores de los vértices.
- Implementar un importador de ficheros PLY propio. Se intentó utilizar la librería ASSIMP, pero daba problemas a la hora de linkear la librería en ciertas ocasiones y pesaba demasiado. Además, se observó que el formato de los ficheros PLY era bastante sencillo, por lo que dicha implementación tuvo escaso impacto de trabajo.
- Limitar el fichero PLY. Puesto que se está programando un proyecto pequeño con muy pocos modelos 3D se puede considerar un entorno controlado, y por tanto limitar las propiedades y formatos de los ficheros PLY para hacerlos más rápidos y fáciles de leer. Para ello se han eliminado las cabeceras dejando únicamente el número de vértices y caras y, a la hora de exportarlos desde blender, se han convertido todas las caras a triángulos.

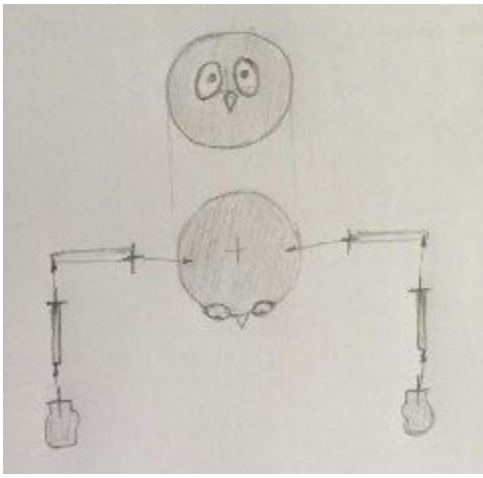
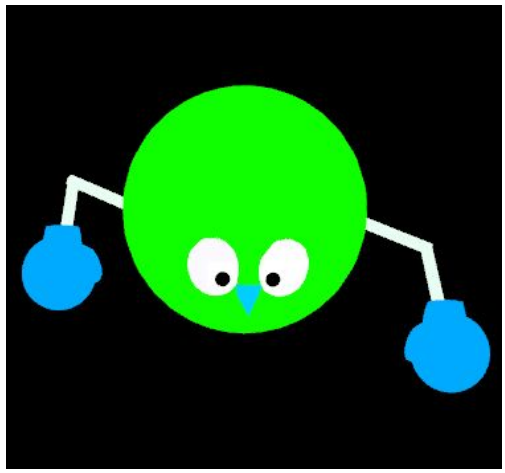
- De cada fichero PLY se leen la posición, normal y color de cada vértice. Posteriormente, se leen las caras para realizar un indexed rendering y se manda todo al pipeline de la GPU .
- Cada parte móvil de cada animación tendrá su propio modelo 3D. De este modo las animaciones en 3D se crean por medio de transformaciones geométricas locales: traslación en los saltos de los SNO-BEES, cizalla cuando los enemigos están paralizados y rotaciones en el resto de los casos. Esto se consigue también estableciendo un sistema de jerarquía entre los componentes, en el que se realizan primero las transformaciones locales (por ejemplo rotar el brazo y colocarlo en el hombro de pengo) y luego las transformaciones del componente 'padre' (mover a pengo hacia abajo).

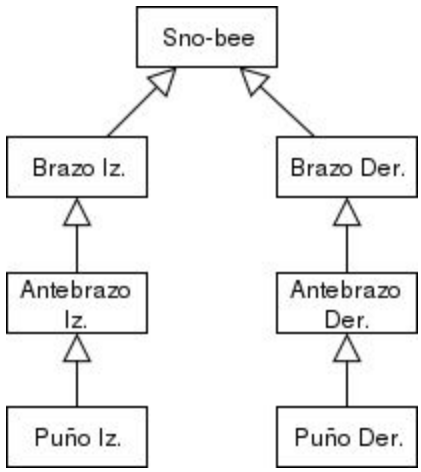
  A continuación se muestran los modelos utilizados:
- PENGO. Se compone de 5 componentes: Cuerpo y cabeza, dos brazos y dos pies.



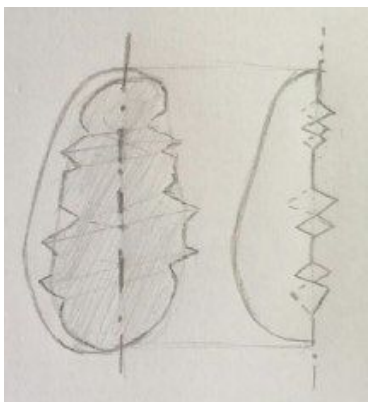
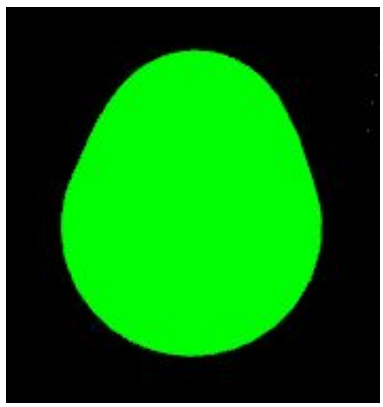
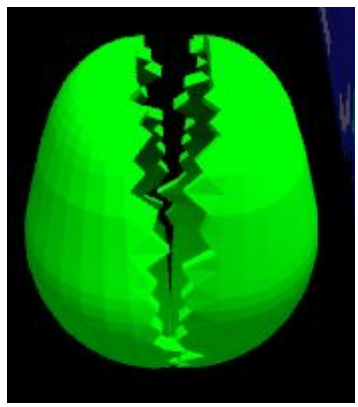


- SNO-BEE. Se compone de 7 componentes: Cabeza, dos brazos, dos antebrazos y dos puños.



- Huevo de SNO-BEE. Se compone únicamente de dos partes: una mitad izquierda y otra derecha. Ambas encajan perfectamente, de modo que sirven para simular tanto el huevo cerrado como abriéndose.

Los bloques de hielo y las paredes se realizaron mediante unas mallas de triángulos que forman un cubo. Por su simplicidad, estas mallas no cuentan con un fichero PLY, sino que están hardcoreadas en código (se especifican manualmente las posiciones de los vértices, sus normales y sus coordenadas UV). Para realizar menos cambios respecto a la versión anterior y mantener la estética, se han reutilizado las mismas texturas y animaciones que en su versión 2D.

Por último, tras la entrega parcial de la versión 3D, se incluyó iluminación a la escena 3D. Para ello únicamente fue necesario modificar los shaders añadiendo el cálculo de la iluminación tal y como se muestra en los tutoriales de openGL. En los modelos complejos únicamente se han aplicado los componentes ambiente y especular, mientras que en los bloques y paredes se aplican ambiente, difuso y especular. Se ha establecido una opción en el menú del juego para activar/desactivar la iluminación.

6. Velocidad de juego (Bucle de juego)

El bucle de juego implementado es la última versión comentada en este artículo:
http://www.koonsolo.com/news/dewitters-gameloop/

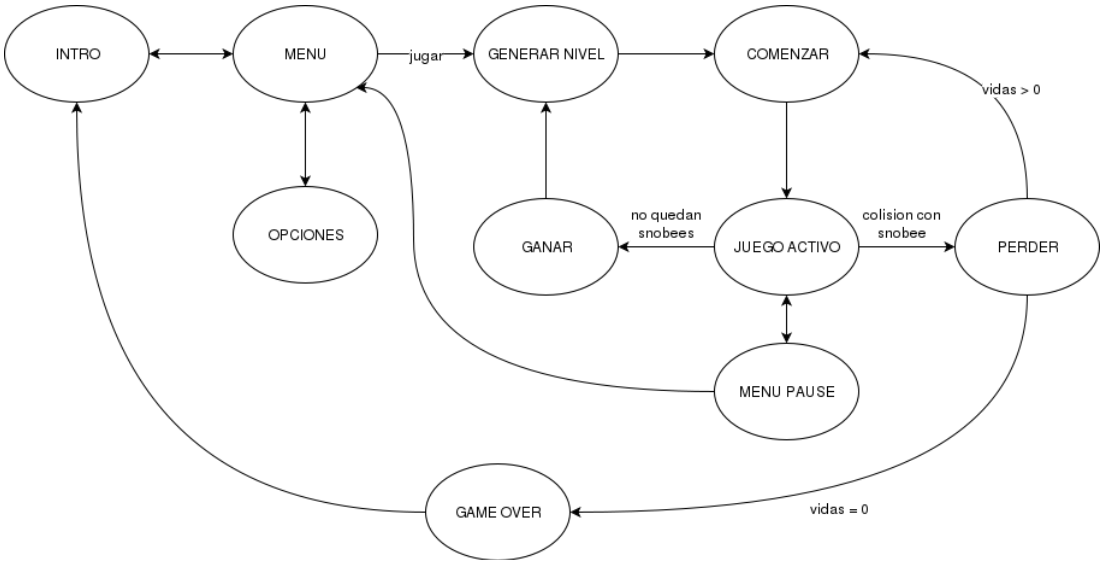
Consiste en llamar a la función update(), en la que se reproduce el sonido y se calculan los movimientos, 25 veces por segundo, limitando la lógica del juego a 25 FPS y utilizar los frames restantes para pintar en pantalla con la función render(). Para ello hay que calcular el momento en el que se llama a la función display() pasándole una interpolación respecto a update() para ajustar el movimiento. En pseudocódigo:

```
const int TICKS_PER_SECOND = 25;
const int SKIP_TICKS = 1000 / TICKS_PER_SECOND;
const int MAX_FRAMESKIP = 5;

DWORD next_game_tick = GetTickCount();
int loops;
float interpolation;

bool game_is_running = true;
while( game_is_running ) {

    loops = 0;
    while( GetTickCount() > next_game_tick && loops < MAX_FRAMESKIP) {
        update_game();

        next_game_tick += SKIP_TICKS;
        loops++;
    }

    interpolation = float( GetTickCount() + SKIP_TICKS - next_game_tick )
                    / float( SKIP_TICKS );
    display_game( interpolation );
}
```

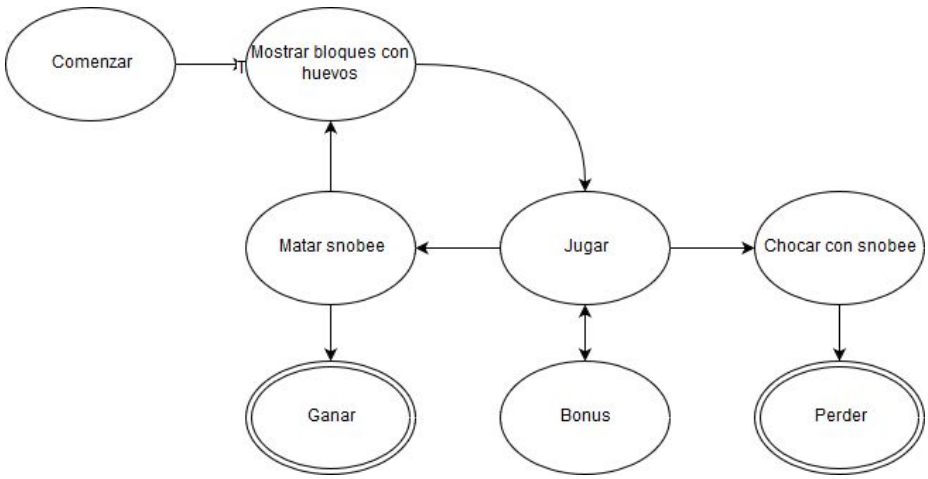
7. Lógica del juego

Algunas de las clases implementadas requieren especial mención por asegurar la lógica del juego. Entre ellas destacan:

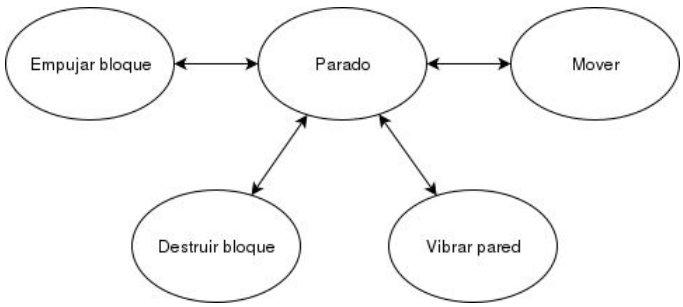
- Game: Mantiene el flujo principal de juego. Implementa la siguiente máquina de estados:



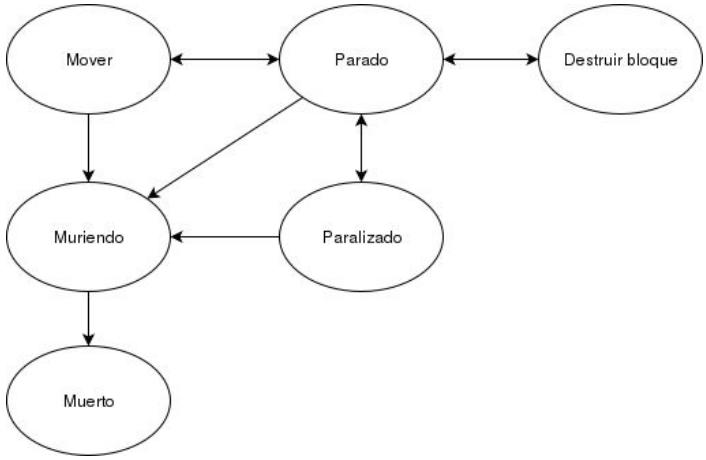
- GameLevel: Mantiene el flujo de juego de una partida. Implementa la siguiente máquina de estados:



- Player: Garantiza que las acciones de PENGO se realizan desde el estado “Parado”:



- Snobee: Implementa la máquina de estados durante la vida de un SNOBEE:

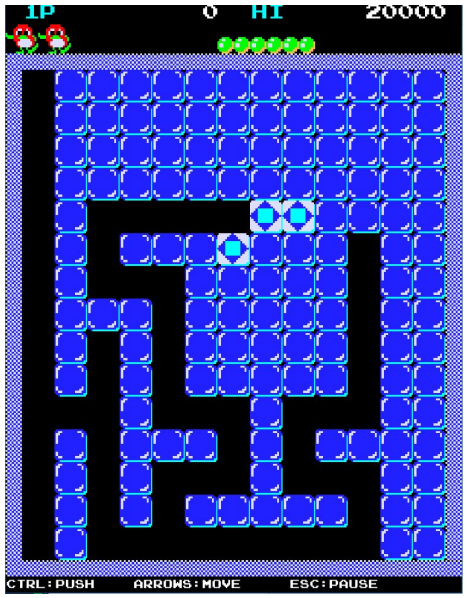


El flujo principal del juego se puede explicar también a través de las siguientes pantallas:

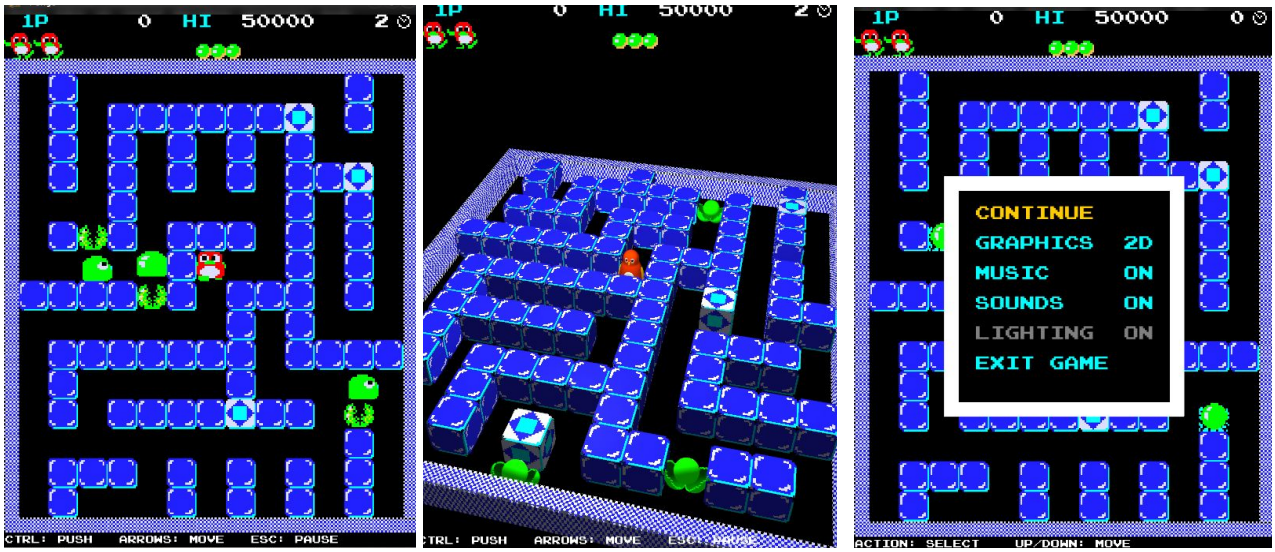
- Intro del juego: Aparece una animación con pingüinos en un paisaje ártico. Tras finalizar automáticamente o por medio del botón de acción del jugador la pantalla cambiará a la de menú start.

- Menú start: Aparece el título del juego donde se puede seleccionar Play para empezar a jugar, Options para modificar opciones de juego y Exit para salir del juego.

- Pantalla de generación de nivel: Aparece la pantalla llena de bloques de hielo que se van desvaneciendo progresivamente recorriendo los pasillos (casillas vacías) hasta dejar a la vista la pantalla de juego.

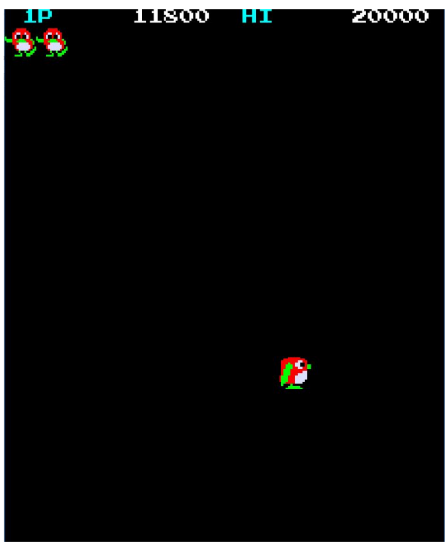


- Pantalla de juego: Pantalla que donde se desarrolla la acción del juego. Al pulsar la tecla ESC se pausa el juego y aparece el menú de opciones, que permite seleccionar modo de juego (entre 2D y 3D), activar/desactivar la música y los sonidos y salir del juego.

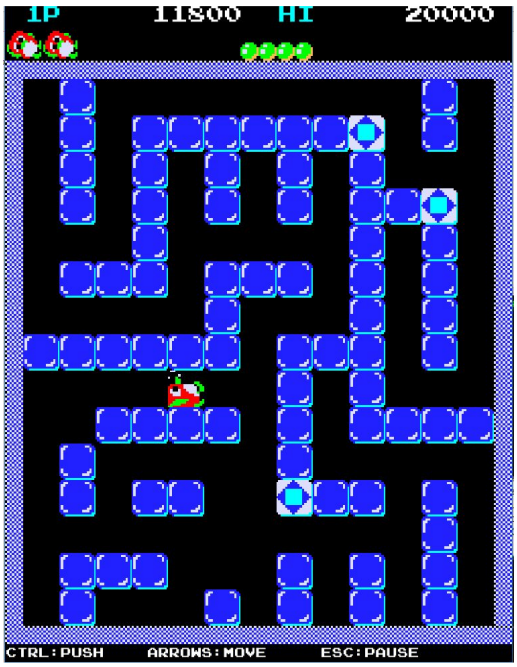


- Pantalla de bonus por diamantes: Aparece cuando se junten los 3 bloques de diamante.

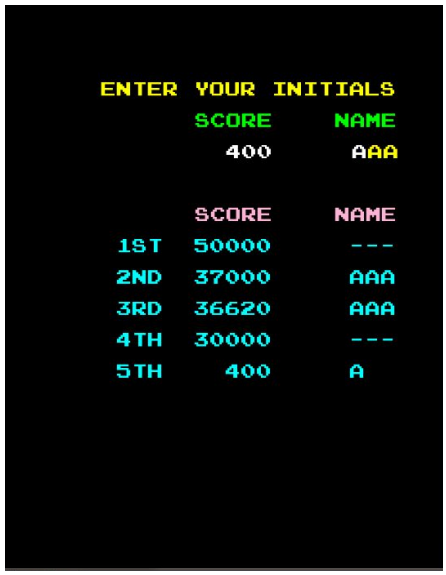
- Pantallas de victoria: Tras acabar con los SNO-BEES desaparecen los bloques de la pantalla y PENGO se dirige a la derecha o a la izquierda dependiendo de lo lejos que esté de ese lateral en el momento de acabar la partida. Después, muestra el tiempo transcurrido en la partida y su bonificación correspondiente. Además, aparecerán distintas animaciones para cada nivel celebrando la victoria. Por último cambia automáticamente a la pantalla de carga del siguiente nivel.



- Pantallas de derrota: Si el jugador pierde la partida PENGO cae sobre su propia espalda. Después aparece una pantalla en negro con el texto “READY?” y vuelve a la pantalla de juego del mismo nivel (sin pasar por la pantalla de generación de nivel). La distribución de los bloques de hielo comienza tal y como quedó antes de la derrota, mientras que PENGO comenzará en el centro y los SNO-BEES en las esquinas del nivel.



- Pantallas de fin del juego: Cuando el jugador pierde todas las vidas aparece un mensaje de “Game over”. Después mostrará una pantalla con tu puntuación y las 5 mejores. Si tu puntuación supera alguna de estas 5 mejores se registrará ingresando un nombre con un máximo de 3 caracteres. Posteriormente aparecerá una pantalla en negro con el texto “THANKS FOR PLAYING” y transcurridos unos segundos comenzará el juego de nuevo desde la pantalla de intro del juego.



8. Jugabilidad y controles

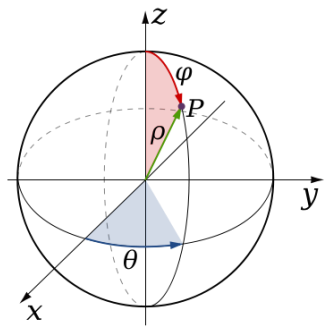
Todo el juego principal funciona únicamente con 6 botones. Por defecto se han asignado los siguientes, aunque se pueden seleccionar otras desde el menú de opciones:

- Flechas direccionales: En los menús permite seleccionar una opción con las flechas arriba y abajo. Durante el juego, permitirá mover a PENGO.
- Tecla CTRL izquierda: En los menús permite modificar una opción o seleccionarla. Durante el juego, permitirá empujar o destruir un bloque que esté en frente de PENGO.
- Tecla ESC: Pausa la partida y abre el menú de opciones durante el juego.

Además se han definido controles específicos para controlar la cámara durante el juego:

- Tabulador: Cambia entre posiciones de cámara prefijadas.
- Desplazar el ratón mientras se pulsa el botón izquierdo: Rota la cámara alrededor de un punto central, por defecto el centro del laberinto.
- Desplazar el ratón mientras se pulsa el botón derecho: Mueve el punto central al que apunta la cámara.
- Rueda del ratón: Aumentar/disminuir zoom.

9. Cámara



En la versión 3D se ha implementado una cámara de tipo arcball, es decir, una cámara que se mueve a lo largo de una semi-esfera situada en el centro del nivel. Por tanto, la posición de la cámara en coordenadas cartesianas se calcula en base a ϕ (ácimo) y φ (altitud). Tanto ϕ como φ se pueden modificar pulsando el botón izquierdo del ratón y desplazándolo en horizontal y vertical respectivamente. También se puede modificar la posición del centro de la esfera en el nivel presionando el botón derecho del ratón y desplazando, así como alterar su radio con la rueda del ratón.

Para que resulte más difícil perderse modificando la cámara se ha decidido limitar la altitud en el rango (0-90], la posición dentro del mapa y el zoom (radio de la semi-esfera). Además, se puede volver a las posiciones pre-guardadas en cualquier momento con el tabulador. Estas posiciones son una primera en vista cenital, simulando la versión en 2D, y una segunda más ladeada.

Por último, se decidió que cuando el juego esté en pause no se muestre el menú correspondiente mientras se modifica la cámara para permitir una mejor visualización de la escena.

10. Generación de niveles

Una de las ideas iniciales fue generar los niveles aleatoriamente, pero se acabó descartando puesto que se consideró que podrían alterar la jugabilidad. Por contraparte, se asume que los niveles originales del juego han sido muy testeados por los diseñadores y cuentan con una buena jugabilidad, por lo que se ha decidido utilizar los mismos y evitar complicaciones innecesarias. De esta forma, los niveles están previamente guardados en ficheros localizados en el directorio “levels”. Estos ficheros están formados por 15 líneas de 13 dígitos separados por espacios, los cuales indican el contenido de las celdas del nivel:

- 0 representa celda libre.
- 1 representa bloque de hielo.
- 2 representa bloque de diamante.

Una vez leído el contenido de dicho fichero se rellena el mapa de bloques, pero también se rellena un mapa auxiliar para simular la generación progresiva del nivel. Esto se hace incluyendo bloques “fantasma” en los pasillos vacíos, que se irán eliminando progresivamente hasta que no quede ninguno en el mapa auxiliar, dejando a la vista el mapa original. El algoritmo de borrado empieza en la esquina inferior izquierda y avanza en alguna de sus posiciones vecinas, y si hay varias opciones se pospone el avance en una cola FIFO hasta que no puede avanzar más. Si también está vacía la cola FIFO se continuará por cualquiera de los bloques restantes.

Una vez generado el nivel, se coloca a PENGO en la posición central del mapa y se reparten los huevos de SNOBEES aleatoriamente entre los bloques de hielo. Posteriormente se abren 3 de ellos al azar y comienza la partida.

11. Inteligencia artificial

La IA de los enemigos ha ido evolucionando desde la creación de los mismos hasta intentar lograr los objetivos mencionados en el GDD. Inicialmente se les asignó un movimiento completamente aleatorio, únicamente para que se movieran mientras se probaban el resto de componentes del juego.

Posteriormente se les limitó a que no volvieran a la posición por la que habían venido, haciendo así que pudieran recorrer pasillos enteros. Esto hacía que fueran muy previsibles, pero parecían más inteligentes. Una vez que se completó el funcionamiento del juego (permitiendo que los SNO-BEES destruyeran bloques, etc.) se intentó entrenar una red neuronal para el movimiento de los SNO-BEES, pero no dio tiempo a tenerla lista para la demo 2D + IA. Por ello, se decidió implementar un algoritmo simple para que los SNO-BEES intentarán rodear a PENGO por varios flancos a la vez generando una dificultad considerable.

Finalmente, para la versión 2D+3D+IA se ha implementado una red neuronal basada en Perceptrón Multicapa que controla el movimiento de cada SNO-BEE:

- 4 neuronas de entrada :
  - Input 1 : Valor de la casilla encima del SNO-BEE.
  - Input 2 : Valor de la casilla a la derecha del SNO-BEE.
  - Input 3 : Valor de la casilla debajo del SNO-BEE.
  - Input 4 : Valor de la casilla a la izquierda del SNO-BEE.

  Los valores que puede tomar cada Input son:
  - -1 si hay pared, bloque de diamante, bloque de hielo con huevo dentro u otro SNO-BEE.
  - Distancia desde esa casilla a PENGO, si hay un bloque de hielo o es una casilla vacía.

- 16 neuronas en la capa oculta divididas en dos capas, 8 en cada capa.

- 2 neuronas de salida [Output1, Output2]:
  - [0, 0] : Moverse ARRIBA.
  - [0,1] : Moverse DERECHA.
  - [1,0] : Moverse ABAJO.
  - [1,1] : Moverse IZQUIERDA

Para entrenarla se han generado 1000 posiciones aleatorias de PENGO y de un SNO-BEE en el campo de juego y se han establecido los valores de las neuronas de entrada según el criterio anterior, siendo la salida deseada para cada conjunto de entradas el movimiento hacia la casilla que tuviera como entrada el valor positivo más pequeño. Esto es que el movimiento esperado sea hacia la casilla libre más cercana a PENGO.

Para establecer el número de neuronas ocultas a utilizar, se hicieron varias pruebas:

1. 8 neuronas en una única capa: El error cuadrático medio era muy alto y el movimiento de los SNO-BEES parecía aleatorio.
2. 16 neuronas en una única capa: El error se redujo pero no significativamente, el movimiento seguía pareciendo aleatorio.
3. 64 neuronas en una única capa: Aquí sí que se redujo el error drásticamente pasando a ser casi 0. Con esta versión no había diferencias con el comportamiento por reglas de perseguir a PENGO.
4. 16 neuronas en 2 capas, 8 en cada capa: Probamos a utilizar más de una capa de neuronas ocultas y el resultado fue bastante bueno, con esta configuración el error obtenido fue menos de la mitad que con las 16 neuronas ocultas en una única capa.

Esta última configuración fue la que presentamos en la demo previa a la entrega final y seguía siendo muy difícil ya que la mayor parte del tiempo los SNO-BEES perseguían a PENGO a pesar de que en alguna ocasión se desviarán por el error cometido por la red.

Finalmente se corrigió esto distinguiendo entre casillas vacías y bloques de hielo normales, si la entrada es un bloque de hielo normal, el valor de esa entrada se aumenta en 2 unidades, de modo que se avanza antes hacia una casilla vacía que romper un bloque de hielo. También se penaliza el volver a la casilla anterior ya que sino se podría entrar en un estado de moverse únicamente entre 2 casillas.

12. Aumento de dificultad

La dificultad del juego aumentará conforme se vayan completando niveles. Para dotar de esa dificultad se modificarán los siguientes factores:

- Al avanzar cada nivel se aumenta la velocidad de los SNO-BEE hasta un máximo.
- Al avanzar de nivel se aumenta el número de huevos de SNO-BEE hasta un máximo.
- Al pasar de cierto nivel, los SNO-BEES pasan de ser 3 a 4.

Esta es la tabla con los parámetros de cada nivel:

NIVEL	Nº SNO-BEES	Nº HUEVOS	VELOCIDAD
1	3	6	0.085
2	3	7	0.090
3	3	8	0.095
4	3	9	0.100
5	3	10	0.105
6	3	11	0.110
7	4	12	0.085
8	4	12	0.090
9	4	12	0.095
10	4	12	0.100
11	4	12	0.105
12	4	12	0.110
13+	4	12	0.11

13. Problemas encontrados y soluciones

- Cuando se obtienen las coordenadas (u,v) desde los vértices se apreció que se encontraban en el rango [0,1], en vez de en el rango [0,1). Por ello, cuando se intentaba obtener un frame de una animación desde una spritesheet se mostraba una fila/columna de más. Se consiguió parchear el problema recortando dicho rango manualmente con un valor pequeño (0.002) y utilizando un filtrado de textura del tipo “vecino más cercano”.
- Algunas de las caras de los modelos 3D no se veían correctamente. Se ha solucionado quitando el culling.
- Problemas de portabilidad de la librería assimp. Se ha solucionado creando nuestro propio cargador de modelos.
- La cámara ortogonal de el menú principal no mostraba correctamente los modelos 3D. Se ha solucionado escalando los modelos en el eje Z (hacia la cámara) a tamaños muy pequeños y cambiando el orden de las transformaciones.
- La IA de los enemigos era demasiado complicada. Se ha solucionado penalizando ciertos movimientos de los SNOBEES (destruir bloques, volver hacia atrás)

14. Tareas restantes

Por motivos de tiempo, han quedado sin implementar algunas características que hubieran resultado interesantes:

- Nueva funcionalidad para el 3D
- Personalizar color de PENGO
- Guardar configuraciones en fichero
- Encriptar rankings
- Animaciones geométricas de las paredes al vibrar
- Animación de destruir bloques de hielo con partículas