**Nombres: Nicolas Pérez Teran, José Daniel Montero Gutiérrez, Kevin David Hernández Tapiero.**

**Documento de diseño**

* **Requerimientos funcionales:**

Se requiere que se cumplan las siguientes funciones:

* **Menú principal:** se requiere que exista un menú que permita al usuario elegir que niveles quiere jugar y cargue todos los elementos del juego (puntuaciones, personalidades de los personajes, sprites, etc).
* **Actualización periódica del tablero de juego con cada interacción realizada por el jugador o la maquina:** se necesita que pueda verse una relación directa entre el tablero y las entradas hechas por el jugador o la máquina.  
  *(Segunda iteración)* Mientras que transcurre la partida, los fantasmas deberán de hacer uso de información como a qué dirección está yendo o qué direcciones tienen disponibles para desplazarse, por lo que estos elementos específicos deben tenerse en cuenta al programar la interacción entre el jugador y su entorno.
* **Interacción entre el jugador y otros objectos del videojuego:** se necesita que haya una garantía de que el jugador pueda ganar puntos, power-ups o perder vidas dependiendo de una variable colisión con otro cuerpo.
* **Conteo de mejores puntajes:** se requiere que se vayan almacenando y actualizando las mejores puntuaciones en una lista de 10 elementos.
* **Cargar tableros de juego:** se requiere que puedan importarse y utilizarse varios tableros para jugar.
* *(Segunda iteración)* **Cronómetro:** cuenta hasta 0 y provocará que el jugador pierda una vida.
* (Segunda iteración) **Alterar posición de Pacman/fantasma:** al pasar a través de un borde sin pared, el jugador aparecerá en el lado opuesto de la pantalla. Esto se puede lograr si al cruzar un borde se cambiara la posición en *x* o *y* (según sea respectivo) por la opuesta en la *matriz* del mapa.
* *(Segunda iteración)* **Cambio de velocidad del jugador:** un tipo de superpoder haría a Pacman más rápido (multiplicando su velocidad por un número entero positivo), permitiéndole escapar más fácilmente de los fantasmas.
* *(Segunda iteración)* **Actualización de la tabla de puntajes:** al finalizar una partida, debe existir una función que revise si el puntaje obtenido en la partida sobrepasa a alguno en el marcador, mover esos puntajes hacia abajo una posición y asignar el puesto del nuevo puntaje.
* **Requerimientos no funcionales:**
* Se requiere que las personalidades de los fantasmas puedan ser fácilmente alteradas o agregar fácilmente nuevas personalidades.
* *(Segunda iteración)* Se requiere establecer la posición de Pacman al lado opuesto del mapa cuando cruza un borde en el que no hay una pared.
* **Restricciones:**
* El programa debe ser desarrollado en Java.
* Se requiere que las actualizaciones sean por medio de un método que reciba la acción a realizar en el tablero –la acción realizada debe ser un parámetro int llamado jugada-.
* Los archivos de tableros deben ser leídos a raíz de un archivo de texto.
* Se requiere que los nuevos datos agregados –por ejemplo, las puntuaciones- se mantengan sin importar las veces que se ejecute el juego.
* **Roles y responsabilidades:**

Inicialmente, debemos saber que el problema se basa en que el jugador debe estar expuesto a un reto que le permita entretenerse y pasar un buen rato. Para esto se requiere que haya variación entre mapas y enemigos, así que el objetivo será ir aumentando la dificultad con los cambios que podemos hacer: variación en la personalidad y comportamiento de enemigos.

* **Controlador:** Será aquella parte encargada de manejar todas las acciones y objetos que interactúen durante la partida.
* **Pacman: S**erá el objeto que el jugador controlara; tiene la capacidad de interactuar con otros cuerpos, ya sea comiendo o perdiendo vidas. Ganará al comer todas las galletas del mapa.
* **Fantasmas:** Antagonistas del juego, tienen como propósito devorar al jugador, además pueden variar sus personalidades.
* **Tablero:** Mapa en el cual se jugará. Aquí estarán presentes e interactuarán todos los elementos del juego.  
  *(Segunda iteración)* Una manera de aumentar la dificultad del juego es exponiendo al jugador a un tablero en el que sus movimientos son más limitados, por lo que se puede hacer uso de este aspecto para añadir al reto del juego.
* **Récord:** Objeto que almacena el nombre del jugador y la puntuación que alcanzo en un tablero.   
  *(Segunda iteración)* Al final de cada partida, la clase *récord* debe tener un método en el que revise si la puntuación obtenida sobrepasa a alguna del top 10 de puntajes registrados.
* **Casilla:** Es cada uno de los espacios que puede ocupar un elemento del juego en el mapa.
* **Objetos de poder:** Permiten que el jugador obtenga un cierto poder temporal.
* **Galletas:** Puntos coleccionables por el jugador, permiten que el jugador gane al desaparecer todos los del tablero.
* *(Segunda iteración)*
* **Poderes:** Puede haber diferentes tipos de poderes que varían su duración y efectos en el personaje.
* **Colaboraciones:**

**El controlador**: tiene una conexión directa con cada uno de los elementos presentes en el juego, pues es aquel que maneja el funcionamiento.

**Jugador:** es quien que funcionará como usuario, tiene la capacidad de manipular a Pacman.

**Pacman:** tiene una relación con los objetos comestibles (galletas y objetos de poder), con el tablero y las casillas –pues en este se mueve y colisiona contra las paredes- y con los enemigos –ya que estos pueden asesinarlo-.

**Fantasmas:** tiene una relación con el tablero y con el jugador, ya que son los antagonistas que se mueven a lo largo del mapa persiguiendo a Pacman.

**Objetos comestibles:** se relacionan con el mapa, la casilla actual, el récord y el jugador, debido a que cada objeto debe estar en una casilla dentro de un tablero especifico y debe poder ser recogida por el jugador para así actualizar el puntaje del récord.

**Récord:** se relaciona con los objetos comestibles y el jugador actual. Recordemos que aquí se almacenan los puntos alcanzados por el jugador.

**Tablero**: se relaciona con todos los elementos y con el controlador –ya que este último es quien se encarga de actualizarlo-.

***Segunda iteracion:***

**Poder:** Se relaciona con el jugador, pues altera ciertos elementos del jugador. Y se relaciona con el tablero, pues ocupa un espacio en él y en una casilla.

**Personalidad:** Se relaciona con el fantasma, pues define su comportamiento.

* **Información que fluye por el sistema:**

En este aspecto, lo más importante serán las colisiones y posiciones de cada uno de los objetos presentes en este sistema. Las posiciones son importantes porque nos permiten hacer un seguimiento al lugar donde se encuentra cada objeto –recordemos que el mapa se basa en casillas donde un individuo puede movilizarse-.

La información manejada de divide en dos grupos: jugadas internas y externas.

* **Jugadas internas:** Son aquellas jugadas que se calculan automáticamente por el sistema. Un ejemplo muy claro serán los comportamientos de los fantasmas, ya que estos manejan una personalidad propia que será derivada de aquello que se encuentre en la información de su código.  
  La interacción que tendrán todos los objetos sale directamente de este tipo de información, pues ya está integrada en cada uno de los objetos.
* **Jugadas externas:** El jugador constantemente interactuara con el sistema a través de los controles del juego, y esto será aquello que permitirá que la posición del jugador vaya cambiando a lo largo del mapa. Esta información de movimiento con dirección será un dato que el jugador nos pasará como parámetro. Con lo anterior, podemos decir que esta sección se limita solo al movimiento del jugador, sin embargo, no es menos importante, recordemos que los cálculos y algoritmos que resultan en jugadas internas son dependientes de la posición que adopte el jugador.
* **Estructuras y grupos de objetos:**  
  Esta sección hace referencia a cómo serán manejados los objetos del juego para su funcionamiento más optimo.

**Controlador/Métodos:**  
**CargarMarcador(archivoMarcador): void.**  
Carga el top 10 de puntajes registrados con sus respectivos nombres.

**CargarTablero(archivoTablero): void.**  
Cargaría el archivo especificado por el jugador para cargar un tablero distinto.

**ObjetoJugador/Pacman: Atributos.**

**Name**: String. Necesitamos que este objeto posea un nombre de usuario –para el récord-

**Vidas**; int. Cantidad de vidas que tiene Pacman, se actualiza cada vez que pierde o gana una de una manera u otra.

**ColisionPared**(): Boolean. Permite saber si el jugador debe seguir moviéndose dependiendo de la colisión que presente con un objeto inamovible.

**PosicionX**: Int .Posición en el eje x del tablero.

**PosicionY:** Int. Posición en el eje y del tablero.

**PuntuacionActual:** Int. Este conteo es bastante importante porque permite saber que récord alcanzo la persona en el tablero.

**DireccionMov:** Int. Se requiere saber hacia qué dirección el jugador se está moviendo para poder hacer su respectiva actualización en la posición en el tablero.

***Segunda iteracion:***  
**Velocidad:** Con algunos poderes el jugador podría ir más rápido, por lo cual debemos manejar cuantos pixeles recorrerá el jugador.

**Invencibilidad:** Con algunos poderes Pacman se volverá inmune al choque con los fantasmas, lo cual significa que no perderá una vida al tocarlos en este estado, sino que ganará puntos y

**ObjetoJugador/Pacman: Métodos.**

**ActualizarPunt(int Punt): Void.** Esta parte se encarga de ir sumando puntos a medida que Pacman come galletas o fantasmas. Recibe un int por parámetro que corresponde al objeto que ingirió el jugador. Esto dependerá del objeto con el cual colisione.

**ActualizarPos(posX, posY):**   
Algo que es necesario es que se actualice la posición con base a lo deseado por el jugador. Esta función se encargará de sumar en la respectiva componente a partir de lo que se dio como parámetro.

**ActualizarVel(nuevaVel):** private void.   
Método que permite alterar la velocidad que el recorrerá los pixeles por una unidad de tiempo.

**ActivarPoder(poder):** void.   
Método que permitirá que el poder agarrado por el jugador este activado.

**DesactivarPoder():** private void.   
Método para devolver los valores iniciales al jugador cuando acabe el tiempo del poder.

**Fantasmas: Atributos.**

**Name: String.** Necesitamos que este objeto posea un nombre para diferenciarlo.

**Personalidad:** Personalidad. Objeto que defina el comportamiento que presentara el fantasma.

**Activo:** Boolean. Al ser devorado, queda inactivo por un periodo de tiempo.

**ColisionPared(): Boolean.** Permite saber si el fantasma debe seguir moviéndose dependiendo de la colisión que presente con un objeto inamovible.

**PosicionX: Int**. Posición en el eje x del tablero.

**PosicionY: Int.** Posición en el eje y del tablero.

**DireccionMov:** **Int.** Se requiere saber hacia qué dirección el fantasma se está moviendo para poder hacer su respectiva actualización en la posición en el tablero.

**Velocidad: Int.** Necesitamos que los fantasmas varíen sus habilidades, en este caso, algunos pueden ser más o menos veloces.

**ObjetoJugador/Pacman: Métodos.**

**ActualizarPos(posX, posY):**   
Algo que es necesario es que se actualice la posición con base a lo dado por el algoritmo del fantasma. Esta función se encargará de sumar en la respectiva componente a partir de lo que se dio como parámetro.

**ActualizarVel(nuevaVel):** private void.   
Método que permite alterar la velocidad que el recorrerá los pixeles por una unidad de tiempo.

**MatarFantasma():** void.   
Cuando un fantasma es asesinado, este vuelve a su lugar de origen y queda desactivado por un tiempo.

**Desac/ActiFan():** public void.   
Cambia el estado activo/desactivo de un fantasma.

**Tablero: Atributos.**

**Nombre: String.** El nombre para diferenciar el tablero de otros.

**Archivo: String.** Archivo de texto de donde se extrae el tablero.

**Tamano: [][].** Tamaño del tablero actual.

**Tablero: Métodos.**

**ActualizarTablero(jugada):** void.  
Se necesita que el tablero se actualice constantemente con cada nueva jugada que se ejecute. Por otro lado, existen elementos con características similares, pero con ligeras diferencias. Esto son los llamados objetos comestibles, que son una generalización de lo que puede comer Pacman.

**Objetos comestibles/Atributos:**

**Name: String.** Todos presentan un nombre con el cual serán identificados.

**PosicionX**: int. Posición en el eje X.

**PosicionY**: int. Posición en el eje Y.

**Objetos comestibles/Métodos:**

**DestruirObj(): void.** Esto es un efecto derivado de la colisión con el jugador.

Pero siguiendo por este aspecto, nos topamos con subclases de estos objetos. Estos son Galletas y Poderes.

* **Galletas/Atributos:**

**PuntosOtorgados: int.** En este atributo se encuentran los puntos que se le otorgaran al jugador cuando consuma este objeto.

**EstaActivo: boolean.** Se encarga de revisar si el objeto fue devorado o no.

* **Galletas/Métodos:**

**ActualizarPuntos(jugador):** Se encarga de verificar si el jugador posee un poder que le permita aumentar la can**t**idad de puntos que recibirá al consumir el objeto.

**RestaurarValorInicial(): void.**  
Se encarga de revertir todos los cambios en los puntos otorgados. Recordemos que el tiempo de un poder es limitado.

**DesaparecerObjeto():** **void**.  
Se encarga de quitar el objeto del juego, recordemos que fue devorado.

**Poderes/Atributos:**  
**Velocidad: byte.** que (de ser distinta a 1) determina cuánto cambia la velocidad del jugador al obtener el poder.  
**Matar: boolean.** que determina si el jugador puede matar fantasmas al obtener el poder.  
**Duración: byte.** que define cuánto dura el poder en segundos.

**Poderes/Métodos:**

**IniciarTemporizador(duracion): void**  
Inicia un temporizador teniendo en cuenta la variable duración (en segundos) que al llegar a 0 acabará con el poder del jugador.

**Récord/Atributos:**  
**Nombre: String.** Tres caracteres que indican el nombre con el que quedará registrado el jugador en caso de estar en el top 10.  
**Puntuación: int.** que define el puntaje que obtuvo el jugador con tal nombre.

**Récord/Métodos:**  
**RevisarPuntaje(puntuacion): void**  
Al finalizar una partida (cuando el jugador haya perdido todas sus vidas), la clase récord revisará si el puntaje hecho por el jugador sobrepasa a alguno en el marcador de top 10 puntajes registrados, de ser así, empujará hacia abajo a los puntajes que estén debajo de él, reorganizando el marcador.

**Fantasma/Atributos:**  
**Personalidad: Personalidad:** Determina cual es el comportamiento de este fantasma.  
**PosicionX: int.** Posicion del fantasma en el eje x   
**PosicionY: int.** Posición del fantasma en el eje x  
**Activo: Boolean.** Verdadero si el fantasma está activo, falso de lo contrario

**Fantasma/Métodos:**  
**MatarJugador(Pacman.vidas, Pacman.PosicionY, Pacman.PosicionX, this. PosicionY, this. PosicionX): void.** Detecta si hay una colisión con el jugador y si este no puede matar al fantasma, el fantasma matará al Jugador

**Personalidad/Atributos:**  
**Nombre: String.** Nombre de la personalidad del fantasma.  
**PatronMovimiento: ArrayList[].** Lista que contendrá comportamientos del fantasma (la distancia a la que debe estar del jugador para que lo persiga, si es más agresivo e incrementa velocidad conforme pasan los niveles, si se alía con el agresivo para acorralar al jugador, o si tiene una personalidad que cambia entre las anteriormente dichas).

**Personalidad/Métodos:**  
**CambiarPersonalidad(): private void.**   
Hace que el fantasma con esta personalidad cambie de patrón de movimiento cada cierta cantidad de segundos.

**Conclusiones finales:**

* Las desventajas es que puede ser un modelo con complicaciones bastante marcadas; muchos de los métodos no alcanzan a ser realmente detallados para que el aprovechamiento de la idea sea el esperado. Además, los poderes que puede obtener son limitados a mejoras temporales.   
  Por otro lado, consideramos que nuestra implementación permite que sea muy flexible a la hora de intercambiar aspectos de los objetos sin problemas; no está sujeto a ningún tipo de elementos, permite que gran parte de las clases estén abiertas a lo que se ponga como parámetro. Un ejemplo claro seria tanto las personalidades de los fantasmas como los mapas posibles –recordemos que ambos pueden ser extraídos de fuentes externas sin problemas mientras se respete el formato del parámetro pedido-.
* Consideramos que algo problemático es la forma en la que se hacen muy pocas especificaciones, lo cual lleva a que enteramente nosotros debamos ir descubriendo los nuevos elementos a agregar; esto es problemático porque puede llevarnos a una posible reestructuración de las ideas cuando previamente no tomamos en cuenda dicha posibilidad y ya hemos trabajado gran parte con dicha idea inicial.