Trabajo Práctico Final

Juego con Clases y Objetos en Visual Studio C++

Trabajo Preliminar

Crear una solución llamada TrabajoPracticoFinal que contendrá el proyecto TPFinal. Deberá completar el código entregado en un archivo .zip.

1° Parte - Movimiento de la nave

Deberá completar el método **input** de la clase **Juego** para mover la nave por la pantalla utilizando las flechas y abandonar el juego presionando la tecla ESC recordando que se debe validar que la nave no supere los limites de la pantalla (los bordes deben ser tenidos en cuenta).
El ancho de la nave es de 5 columnas y el alto de 3 filas.
Abandonar el juego significa activar _gameOver.

```
void input(){
	if(_tecla=getKey(true)){
		switch(_tecla){
			case KEY_UP:  // mueve la nave hacia arriba
				break;
			case KEY_LEFT: // mueve la nave hacia la izquierda
				break;
			case KEY_RIGHT: // mueve la nave hacia la derecha
				break;
			case KEY_DOWN: // mueve la nave hacia abajo
				break;
			case KEY_ESC:{
				break;
			}
		}
	}
}
```

Compile, linkee y pruebe el ejercicio tal que muestre lo siguiente en pantalla:

2° Parte - Creación de los asteroides

Debera crear un array de Asteroides en clase Juego para contenerlos llamado _vecAsT
Los asteroides se crearan en el método **init** y se muestran en el método **draw.**
Deberá completar el método **mover** de la clase **Asteroide** teniendo en cuenta que la validacion incluye cambiar al azar la columna al llegar al final de la pantalla.
En el archivo definiciones.h están MIN_Y, MAX_Y y MAX_X que se deberán utilizar en la validacion del movimiento del asteroide.

```
void Asteroide::mover(){
        borrar();
        borrar();
        _y++;
        // verifica que si la fila supera el limite de la pantalla
        // mueve _x al azar e inicializa y en MIN_Y

        }
}
```

Se crearan 3 asteroides en las coordenadas (10,4), (4,8) y (15,10).
Compile, linkee y pruebe el ejercicio tal que muestre lo siguiente en pantalla:

Deberá crear un atributo estático **_cantAsteroides** tal que lleve la cuenta de los asteroides existentes. Crear el método **getCantAsteroides** y usarlo en el método **display** de la clase **Juego** para que muestre en la columna 55 fila 1 los siguiente:

3° Parte – Colision Asteroide-Nave

Para detectar la colision entre un asteroide y la nave debe completar el método **update** de la clase **Juego** recorriendo el array de asteroides, moviendolos y llamando al metodo colision.
Al momento de la colision se pierde un corazon y al llegar a 0 se pierde una vida
Compile, linkee y pruebe el ejercicio tal que muestre lo siguiente en pantalla:

EXTRA 4° Parte - Creación de las balas

Debera crear un array de **Balas** _vecB en la clase J**uego** para contenerlos.
Las balas se crearan cada vez que se presione la barra espaciadora en el metodo **input** de la clase Juego como se indica a continuacion:

```
        if(_tecla== FIRE){
                for(int i=0;i<TOPE;i++){
                        if(_vecB[i]==NULL)
// crear una bala con la posicion _x de la nave + 2 y la posicion _y de la nave - 1
                break;
         }
```

y se mostraran en el método **draw()**

```
        for(int i=0;i<TOPEB;i++){
                if(_vecB[i]!=NULL)
                // si la bala se encuentra afuera de la pantalla se debe borrar
                // sino se debe dibujar

        }
```

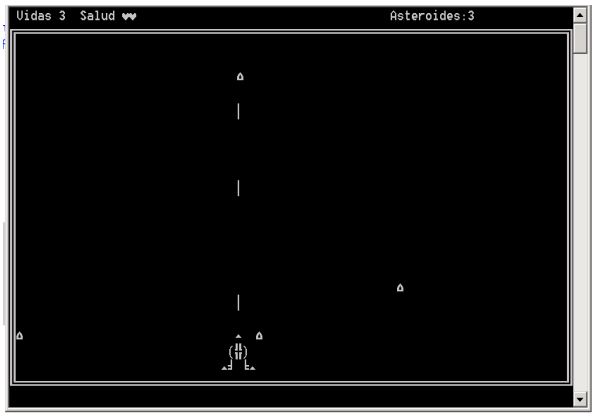
En el método **update** ahora se deben eliminar las balas que no chocan con los asteroides(salen de la pantalla) con el siguiente código:

```
        for(int i=0;i<TOPEB;i++){
                if(_vecB[i]!=NULL){
                        if(_vecB[i]->afuera()){
                                _vecB[i]->borrar();
                                delete _vecB[i];
                                _vecB[i]=NULL;
                        }
                }
        }
```

Para finalizar solo resta la lógica de la colision asteroide-bala también dentro del método **update** de la clase **Juego** para lo cual se deberá : 4
Recorrer el array de asteroides y dentro recorrer el array de balas verificando que la columna del asteroide _x sea la misma que la bala y que la fila del asteroide _y+1 sea igual a la de la bala o que sean iguales las filas de ambos.
Se debe borrar la bala y eliminar ambos bala y asteroide de sus arrays
Por último crear otro asteroide con la siguiente instrucción**:**

```
_vecAst[i]=new Asteroide(rand()%MAX_X-1,MIN_Y);
```

Compile, linkee y pruebe el ejercicio tal que muestre lo siguiente en pantalla:



Veamos las partes del ciclo de un juego:

• Setup. Implica los ajustes iniciales del juego, como el sonido, música y gráficos. También puede presentarse el trasfondo del juego y sus objetivos.
• Ingreso del jugador. Se trata de la captura del ingreso del jugador que puede provenir desde el teclado, mouse, joystick, trackball, o algún otro dispositivo de entrada.
• Actualización del juego. Implica la lógica del juego y las reglas que se aplican al mundo del juego, teniendo en cuenta los aportes del jugador. Puede tomar la forma de un sistema interacción física de objetos o implicar cálculos de Inteligencia Artificial del enemigo.
• Actualización de la pantalla. En la mayoría de los juegos, este proceso es el más exigente del hardware del equipo, ya que a menudo implica gráficos complejos. Sin embargo, este proceso puede ser tan simple como mostrar una línea de texto.
• Comprobación de que el juego ha finalizado. Si el juego no ha terminado (si jugador todavía está vivo y que no ha salido), el control se bifurca de nuevo a la etapa de ingreso del jugador. Si el juego ha terminado, el control se desvía a la fase de cierre.
•Apagado. En este punto, el juego termina. El jugador ve a menudo la información final como ser el puntaje, tiempo, etc. El programa libera todos los recursos, si es necesario, y se cierra.