# Práctica 1: Arrows 1.0

Curso 2019-2020. Tecnología de la Programación de Videojuegos 1. UCM

Fecha de entrega: 1 de noviembre de 2019

El objetivo de esta práctica es implementar en C++/SDL un videojuego sencillo mono-jugador que consiste en disparar flechas mediante un arco con el objetivo de explotar globos que suben por la pantalla. En la actualidad hay multitud de versiones de videojuegos similares, tanto para tablets y móviles, como para PC. Seguramente el primer videojuego de este estilo fue el clásico *Bow & Arrow* de Windows 3.X allá por el año 1992 (ver por ejemplo el vídeo de demostración en https://youtu.be/dcv-XKmUPjo).

A continuación se explica la funcionalidad básica del juego que se debe desarrollar:

- En la mitad derecha de la ventana van apareciendo globos de distintos colores por la parte inferior, que van subiendo (a distintas velocidades) hasta desaparecer por la parte superior.
- En la parte izquierda de la ventana se visualiza un arco apuntando hacia la derecha controlado por el jugador, que se puede mover en vertical, y con el cual se pueden lanzar flechas. Con los cursores de arriba/abajo se controla el movimiento vertical del arco, con el cursor izquierdo se carga una flecha (si quedan flechas disponibles), y con el cursor derecho se lanza la flecha que esté cargada.
- Una vez lanzada una flecha, ésta se mueve en horizontal de izquierda a derecha a velocidad constante, hasta desaparecer de la ventana por la derecha. Si la punta de la flecha impacta con un globo, éste explota (con la correspondiente animación usando las diferentes texturas proporcionadas), genera una puntuación que se suma a la puntuación acumulada de la partida, y la flecha prosigue su camino, pudiendo explotar más globos.
- Tanto la puntuación como el número de flechas disponibles deben mostrarse de alguna manera y actualizarse correspondientemente (al menos en la consola, y opcionalmente, en la ventana gráfica).
- El juego acaba cuando se acaban las flechas disponibles, inicialmente 10.

# Detalles de implementación

## Diseño de clases

A continuación se indican las clases y métodos que debes implementar obligatoriamente. A algunos métodos se les da un nombre específico para poder referirnos a ellos en otras partes del texto. Deberás implementar además los métodos (y posiblemente las clases) adicionales que consideres necesario para mejorar la claridad y reusabilidad del código. Cada clase se corresponderá con la definición de un módulo C++ con sus correspondientes ficheros .h y .cpp.

**Clase Texture:** Permite manejar texturas SDL. Contiene al menos un puntero a la textura SDL e información sobre su tamaño total y los tamaños de sus frames. Implementa métodos para construir/cargar la textura de un fichero, para dibujarla en la posición proporcionada, bien en su totalidad (método `render`) o bien uno de sus frames (método `renderFrame`), y para destruirla/liberarla. Se darán detalles en clase.

**Clase Vector2D (y Point2D):** Representa vectores o puntos en dos dimensiones y por tanto incluye dos atributos (`x` e `y`) de tipo `double`. Implementa, además de la constructora, métodos para consultar las componentes `x` e `y`, para normalizar el vector, y para la suma, resta, producto escalar de vectores y producto de un vector por un escalar. La suma, resta y productos deben implementarse como operadores. En el mismo módulo define un alias para el tipo `Point2D`.

**Clase Arrow:** Representa a las flechas del juego, tanto a la flecha estática cargada en el arco, como a las flechas en movimiento. Contiene al menos la posición (esquina superior izquierda) como `Point2D`, el ancho y el alto, la dirección/velocidad (`Vector2D`), y un puntero a la textura de la flecha. Implementa, además de la constructura, métodos para dibujarse (`render`), para obtener el rectángulo correspondiente a la punta de la flecha, para establecer la dirección/velocidad, y para actualizarse, moviéndose acorde a su dirección/velocidad (`update`). Este último puede devolver un booleano para indicarle al juego cuando una flecha debe desaparecer de la escena.

**Clase Bow:** Representa al arco del juego. Contiene al menos la posición (esquina superior izquierda) como `Point2D`, el ancho y el alto, la dirección/velocidad (`Vector2D`), un puntero a la flecha cargada (que será `nullptr` en caso de no haber flecha cargada), un puntero a su textura, y un puntero al juego (para informarle cuando se lanza la flecha, la cual pasará como parámetro en la llamada correspondiente). Implementa, además de la constructura, métodos para dibujarse (`render`), actualizarse, moviéndose acorde a su dirección/velocidad (`update`), y para tratar los eventos (`handleEvents`) que controlan el movimiento vertical (cursores arriba y abajo), la carga de una nueva flecha (cursor izquierdo), y el lanzamiento de la flecha cargada (cursor derecho). Es recomendable incluir también un atributo para guardar el estado del arco (cargado/no-cargado) que se utilizaría para distinguir el caso tanto en el movimiento como en la captura de eventos.

**Clase Balloon:** Representa a los globos del juego. Contiene al menos la posición (esquina superior izquierda) como `Point2D` del globo, el ancho y el alto, la dirección/velocidad (`Vector2D`), el estado del globo (normal o pinchado), el instante temporal en el que se pinchó (para controlar la animación del pinchado), un puntero a su textura, y un puntero al juego (para preguntarle si alguna punta de flecha le está tocando, en cuyo caso debe cambiar al estado de pinchado). Implementa, además de la constructura, métodos para dibujarse (`render`) y actualizarse (`update`), moviéndose acorde a su dirección/velocidad. Este último puede devolver un booleano para indicarle al juego cuando un globo debe desaparecer de la escena (bien porque ya se ha pinchado y ha pasado el tiempo de la animación, o bien porque sale por la parte superior de la ventana).

**Clase Game:** Contiene, al menos, punteros a la ventana y al *renderer* de SDL, el booleano `exit`, la puntuación actual, las flechas disponibles, un array con los punteros a todas las texturas del juego, y los (punteros a los) objetos del juego, estando las flechas lanzadas y los globos agrupados en sendos vectores (tipo `vector`). Define también las constantes que sean necesarias (dimensiones de la ventana y de los objetos del juego, `FRAME_RATE` para controlar la velocidad del juego, velocidad de movimiento por defecto de flechas, globos y arco, etc.). Implementa métodos para inicializarse y destruirse, el método `run` con el bucle principal del juego, métodos para dibujar el estado actual del juego (método `render`), actualizar (método `update`), manejar eventos (método `handleEvents`), un método para determinar si dado un globo alguna punta de flecha choca con él (que invocará el propio globo al actualizarse), y finalmente, un método para lanzar una nueva flecha (que invocará el propio arco al manejar el evento correspondiente). También es recomendable implementar un método privado que se encargue de la generación de los globos (que se llamaría desde el método `update`).

### Carga de texturas

Las texturas con las imágenes del juego deben cargarse durante la inicialización del juego (en la constructora de `Game`) y guardarse en un array estático (de `NUM_TEXTURES` elementos de tipo `Texture*`) cuyos índices serán valores de un tipo enumerado (`TextureName`) con los nombres de las distintas texturas. Los nombres de los ficheros de imágenes y los números de frames en horizontal y vertical, deben estar definidos (mediante inicialización homogénea) en un array constante de `NUM_TEXTURES` estructuras en `Game.h`. Esto permite automatizar el proceso de carga de texturas.

### Renderizado del juego

En el bucle principal del juego (método `run`) se invoca al método del juego `render`, el cual simplemente delega el renderizado a los diferentes objetos del juego (globos, flechas, arco, etc.) llamando a sus respectivos métodos `render`, y finalmente presenta la escena en la pantalla (llamada a la función `SDL_RenderPresent`). Cada objeto del juego sabe cómo pintarse, en concreto, conoce su posición y tamaño, y contiene un puntero a su textura asociada. Por lo tanto, construirá el rectángulo de destino e invocará al método `render` o `renderFrame` de la textura correspondiente, quién realizará el renderizado real (llamada a la función `SDL_RenderCopy`).

### Movimientos y manejo de colisiones

Los movimientos de los objetos del juego los desencadena el juego y se van delegando de la siguiente forma: el método `update` del juego (invocado desde el bucle principal del método `run`), va llamando a los métodos `update` de cada uno de los elementos del juego (arco, flechas y globos). Son ellos los que conocen dónde se encuentran y cómo deben moverse. Para saber si un movimiento concreto en una dirección se puede realizar o no, o si provoca alguna acción sobre el estado del juego, deben conocer su *entorno*, es decir, deben preguntar al juego, bien consultando alguna constante o bien llamando a métodos. En particular, cuando un globo se mueve, éste le preguntará al juego si la punta de alguna flecha le toca, en cuyo caso el globo cambiará de estado.

### Manejo básico de errores

Debes usar excepciones de tipo `string` (con un mensaje informativo del error correspondiente) para los errores básicos que pueda haber. En concreto, es obligatorio contemplar los siguientes tipos de errores: fichero de imagen no encontrado o no válido, y error de SDL. Puesto que en principio son todos ellos errores irrecuperables, la excepción correspondiente llegará hasta el `main`, donde deberá capturarse, informando al usuario con el mensaje de la excepción antes de cerrar la aplicación.

## Pautas generales obligatorias

A continuación se indican algunas pautas generales que vuestro código debe seguir:

- Asegúrate de que el programa no deje basura. Para que Visual te informe debes escribir al principio de la función `main` esta instrucción

  ```
  _CrtSetDbgFlag( _CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF );
  ```

  Para obtener información más detallada y legible debes incluir en todos los módulos el fichero `checkML.h` (disponible en la plantilla en el proyecto `HolaSDL`).
- Todos los atributos deben ser privados excepto quizás algunas constantes del juego en caso de que se definan como atributos estáticos.

- Define las constantes que sean necesarias. En general, no deben aparecer literales que pudiesen corresponder con configuraciones del programa en el código.
- No debe haber métodos que superen las 30-40 líneas de código.
- Escribe comentarios en el código, al menos uno por cada método que explique de forma clara qué hace el método. Sé cuidadoso también con los nombres que eliges para variables, parámetros, atributos y métodos. Es importante que denoten realmente lo que son o hacen. Preferiblemente usa nombres en inglés.

# Funcionalidades opcionales (2 puntos adicionales máximo)

- Permitir controlar la velocidad de las flechas al lanzarse, por ejemplo dependiendo del tiempo transcurrido desde la carga de la flecha a su lanzamiento.
- Permitir modificar la orientación del arco (y la flecha cargada) y por tanto también de los disparos de las flechas. Puedes utilizar para ello los parámetros de rotación (`angle` y `center`) del método `SDL_RenderCopyEx`.
- Mostrar en la ventana gráfica la puntuación y las flechas disponibles. Para ello debes definir una nueva clase `ScoreBoard` con al menos su método `render`.
- Mantener un registro con las mejores 10 puntuaciones (y nombres de jugadores asociados) en partidas pasadas y actualizarlo con cada partida acabada cuya puntuación entre en el *top-10*. Ver el ejercicio 5 del tema 2.
- Implementar el soporte para permitir guardar y cargar partidas.

# Entrega

En la tarea del campus virtual *Entrega de la práctica 1* y dentro de la fecha límite (1 de noviembre), cada uno de los miembros del grupo, debe subir un fichero comprimido (.zip) que contenga la carpeta de la solución y el proyecto limpio de archivos temporales (asegúrate de borrar la carpeta oculta .vs y ejecuta en Visual Studio la opción "limpiar solución" antes de generar el .zip). La carpeta debe incluir un archivo `info.txt` con los nombres de los componentes del grupo y unas líneas explicando las funcionalidades opcionales incluidas y/o las cosas que no estén funcionando correctamente. Ese mismo texto debes subirlo también en el cuadro de texto (sección "texto en línea") asociado a la entrega.

Además, para que la práctica se considere entregada, deberá pasarse una *entrevista* en la que el profesor comprobará, con los dos autores de la práctica, su funcionamiento en ejecución, y si es correcto realizará preguntas (posiblemente individuales) sobre la implementación. Las entrevistas se realizarán en las dos sesiones de laboratorio siguientes a la fecha de entrega, o si fuese necesario en horario de tutorías.

**Entrega intermedia el 15 de octubre:** Un 20 % de la nota se obtendrá mediante una entrega intermedia, que tendrá lugar en la sesión de laboratorio del día 15 de octubre, en la que el profesor revisará el estado actual de vuestra práctica. En particular, se espera que al menos se visualice y mueva el arco mediante los cursores, y que se generen, visualicen y muevan los globos.