Developer Group: OD GAMES

Propuesta de Inversión en Desarrollo de Videojuego

El grupo de desarrollo OD-GAMES realizó un estudio de las diferentes tendencias en el mercado actual de juegos de mesa y videojuegos, en el cual se observó las características que tienen en común los más populares y exitosos, dando como resultado una propuesta para optimizar uno de los juegos mas famosos y divertitdos de todos los tiempos y a su vez presentar una gran oferta de inversión.

Words Battle toma como base la escencia del juego de mesa Scrabble, permitiendo el clásico estilo de multijugador y a su vez competir contra un computador a un cierto nivel de dificultad, el cual puede aumentar de acuerdo al desempeño del jugador (complejidad y puntaje de palabras formadas). De igual forma, rompe con la barrera de la edad, ya que, puede ser jugados por niños. Esto gracias a los algoritmos, bases de datos e inteligencia artifical que posee el juego, los cuales permiten determinar el tipo de palabras que puede conocer un niño a cierta edad.

Los usuarios tendrán la oportunidad de crear un player, guardar partidas y lo mas importante, aprenderan nuevas palabras de una forma divertida. Words Battle cuenta con gráficos de ultima generación gracias a la implementación de poderosas bibliotecas, haciendo asi, una experiencia llamativa y atractiva a los usuarios.

Daniel Figueroa github: danielfigueroad

Oscar Pulido github: Leamak

ENTREGA 1:

Para esta primera entrega se crearon las clases window.h, rectangle.h y token.h, las cuales representaran la ventana, los recatangulos en los cuales se cargarán las imagenes y las fichas (tokens) que a su vez heredaran del tipo rectangle y almacenarán un dato tipo char para representar la letra de la imagen y otro tipo entero para representar su valor en el juego;

La principal herramienta utilizada para la realización de este video juego es la biblioteca Simple Directmedia Layer, la cual, es una biblioteca de desarrollo de software multiplataforma diseñada para proporcionar una capa de abstracción de hardware para componentes multimedia de una computadora.

Esta biblioteca nos permite manejar no sólo hardware, sino también audio, video, fuentes y efectos, los cuales serán agregados para la próxima entrega. De igual forma implementaremos un arbol tipo **MtreeNode** para simular la inteligencia artificial que se encarga de competir e interactuar mediante un chat contra el jugador.

Tambien contamos con un archivo de mas de 80 mil palabras el cual le permitirá al arbol validar cada palabra formada en el tablero y asi calcular los puntajes.

El tipo **TreeSet** se utilizará para mapear las coordenadas del tablero a un valor específico, de esta forma al momento de asignar una ficha a una casilla, sabremos si la ficha está sobre un bonus (por ejemplo: palabra vale x 2, letra vale x 2, letra vale x 3... etc.).

En los directorios se encuentra la carpeta /data, la cual contendrá los usuarios creados y los datos guardados como los nickname, highscore y time. Estos datos pasaran por un algoritmo de cifrado para que solamente sean visibles dentro del juego y no se puedan modificar desde la carpeta.

El equipo tiene proyectado tener listo estos prototipos de datos para la proxima semana.

A continuacion una descripcion de las estructuras ya diseñadas y probadas. Algunas funciones imprimen valores en la terminal con el propósito de hacer tests.

window.h

```
 7 class Window
 8 {
 9   public:
10
11     //La ventana recibe como parámetros un nombre, el ancho y el largo)
12     Window(const std::string &, int, int);
13
14     ~Window();
15
16     //Funcion para inicializar SDL y cargar datos
17     bool init();
18
19     //Funcion para evaluar eventos como movimientos de mouse, presión de teclas, etc;
20     void events();
21
22     //Determina si la ventana esta abierta. Se utilizar para el loop del juego
23     inline bool is_closed() const { return closed; }
24
25     //Renderizador que se encargará de cargar las imagenes al programa.
26     //Debe haber un solo renderizador en el programa por lo cual lo hacemos static
27     static SDL_Renderer * renderer;
28
29   private:
30
31
32     //Nombre de la ventana
33     std::string title;
34
35     //Ancho de la ventana
36     int width;
37
38     //Alto de la ventana
39     int height;
40
41     //Cerrada(true) y abierta(false)
42     bool closed;
43
44     //Ventana del programa tipo SDL_Window
45     //Sus parámetros son:
46     //SDL_Window* SDL_CreateWindow(const char* title, int x_position[CENTERED,UNDEFINED],
47     //int y_position [CENTERED,UNDEFINED],
48     //int width, int height, Uint32 flags[FULLSCREEN/RESIZEABLE/UNDEFINED])
49
50     SDL_Window * window = nullptr;
51 };
```

rectangle.h

```
 3 #include <iostream>
 4 #include <string>
 5 #include <SDL2/SDL.h>
 6 #include <SDL2/SDL_image.h>
 7 #include "window.h"
 8
 9 class Rectangle :
10 {
11   public:
12
13     //Recibe como parametros un renderizador, coord x, coord y
14     //largo y la ruta a la imagen que va a contener
15     Rectangle(const SDL_Renderer *, int , int , int , int , st
16
17     //Evalua los eventos del teclado y mouse
18     void events(SDL_Event &);
19
20     //Dibuja en pantalla el rectangulo
21     void draw() const;
22
23     ~Rectangle();
24
25   private:
26
27     //Rectangulo SDL
28     SDL_Rect rect;
29
30     int widht;
31
32     int height;
33
34     int y_pos;
35
36     int x_pos;
37
38     std::string path;
39
40     //Superficie SDL sobre la cual se renderiza un imagen
41     SDL_Surface * surface = nullptr;
42
43     //Textura SDL, es el dato que contiene la imagen
44     SDL_Texture * texture = nullptr;
45 };
46
47 #endif
```

window.cpp

```cpp
#include <window.h>

SDL_Renderer * Window::renderer = nullptr;

//Asigno los valores a los atributos
Window::Window(const std::string & t, int w, int h) :
title(t), width(w), height(h)
{
  //Si la funcion de inicialización se ejecuta sin ningun error entoces
  //la ventana no esta cerrada
  closed = !init();
}

Window::~Window()
{
  //Destruyo la ventana y cierro SDL
  SDL_DestroyWindow(window);
  SDL_Quit();
}

bool Window::init()
{
  //Este es un dato de control para determinar la biblioteca funciona
  if (SDL_Init(SDL_INIT_VIDEO) != 0)
  {
    std::cerr << "Failed to initialize SDL!\n";
    return false;
  }

  //Creo la ventana con un nombre, centrada en x e y, y asigno las dimensiones
  else
  {
    window = SDL_CreateWindow(title.c_str(), SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED, width, height, 0);

     //Si falló la creacion de la ventana
    if (window == nullptr)
    {
      std::cerr << "Failed to create window!\n";
      return false;
    }

    //Creo el renderizador
    renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED);
```

C++

```cpp
    //Este es otro dato de control para ver si las bibliotecas libpng, libjpg... funcionan correctamente
    if (IMG_Init(IMG_INIT_PNG) != IMG_INIT_PNG)
    {
      std::cerr << "Failed to initialize SDL Image" << '\n';
      return false;
    }

    return true;
  }
}

void Window::events(SDL_Event & event)
{ //Evaluo el evento capturado
  switch(event.type)
  {
    //Si presiono la (x) para cerrar la ventana
    case SDL_QUIT:
      //Se cierra la ventana
      closed = true;
      break;

    case SDL_MOUSEMOTION:
      //Muestra por terminal las coordenadas x, y del mouse
      //Esto se utilizará mas adelante para mover las fichas con el mouse y no el teclado
      std::cout << event.motion.x << ", " << event.motion.y << '\n';
      break;

    case SDL_MOUSEBUTTONUP:
    //Si presiono o suelto el boton del mouse, imprimelo por terminal
      std::cout << "Mouse button released\n";
      break;

    default:
      break;
  }
}


void Window::clear(SDL_Renderer * renderer) const
{
  //Esta funcion presenta el renderizador
  //Asigna un color para rellenar la ventana (en caso de que no haya una imagen)
  //Y limpia el renderizador
```

rectangle.cpp

rectangle.cpp (~/Desktop/PROYECTO_PR3/src) - gedit

Open

rectangle.cpp

```
#include <rectangle.h>

//Asigno los valores a los atributos
Rect::Rect(int w, int h, int x, int y, std::string p) :
width(w), height(h), x_pos(x), y_pos(y), path(p)
{
  //Cargo la imagen a la superficie
  surface = IMG_Load(path.c_str());
    |
  if (!surface) //Si la carga no fue exitosa entonces
    std::cerr <<"Failed to load surface\n";

  //De lo contrario crea la textura de la superficie
  texture = SDL_CreateTextureFromSurface(Window::renderer,surface);

  if (!texture) //Si la creaciónno fue exitosa entonces
    std::cerr<< "Failed to create texture\n";

  //Y libero la superficie
  SDL_FreeSurface(surface);
}

Rect::void events(SDL_Event & event)
{
  //Si el evento capturado es la presion de una tecla
  if (event.type == SDL_KEYDOWN)
  {  //Que tecla se presionó?
    switch (event.key.keysym.sym)
    {
        //Flecha izquiera : mueve el rectangulo (la ficha en nuestro caso) a la izquiera de pixel en pixel
       case SDLK_LEFT:
         x_pos -= 1;
         break;
        //Flecha derecha: mueve la ficha a la derecha
       case SDLK_RIGHT:
         x_pos += 1;
         break;
        //Flecha arriba : mueve la ficha hacia arriba
       case SDLK_UP:
         y_pos -= 1;
         break;
        //Flecha abajo : mueve la ficha hacia abajo
       case SDLK_DOWN:
```

rectangle.cpp

```
31        case SDLK_LEFT:
32          x_pos -= 1;
33          break;
34         //Flecha derecha: mueve la ficha a la derecha
35        case SDLK_RIGHT:
36          x_pos += 1;
37          break;
38         //Flecha arriba : mueve la ficha hacia arriba
39        case SDLK_UP:
40          y_pos -= 1;
41          break;
42         //Flecha abajo : mueve la ficha hacia abajo
43        case SDLK_DOWN:
44          y_pos += 1;
45          break;
46     }
47
48   }
49 }
50
51 Rect::void draw() const
52 {
53   //Asigno las dimensiones al rectangulo
54   SDL_Rect rect = {x_pos, y_pos, wid, hei};
55
56   if (texture) //Si la textura no es nullptr, la cargo al rectangulo
57     SDL_RenderCopy(Window::renderer,texture, nullptr, & rect);
58
59   //De lo contrario relleno el rectangulo con colores r, g, b, a y lo paso al renderizador
60   else
61   {
62     SDL_SetRenderDrawColor(Window::renderer,0,0,200,255);
63
64     SDL_RenderFillRect(Window::renderer, rect);
65   }
66 }
67
68 Rect::~Rect()
69 {
70   //Destruyo la textura
71   SDL_DestroyTexture(texture);
72 }
73
```

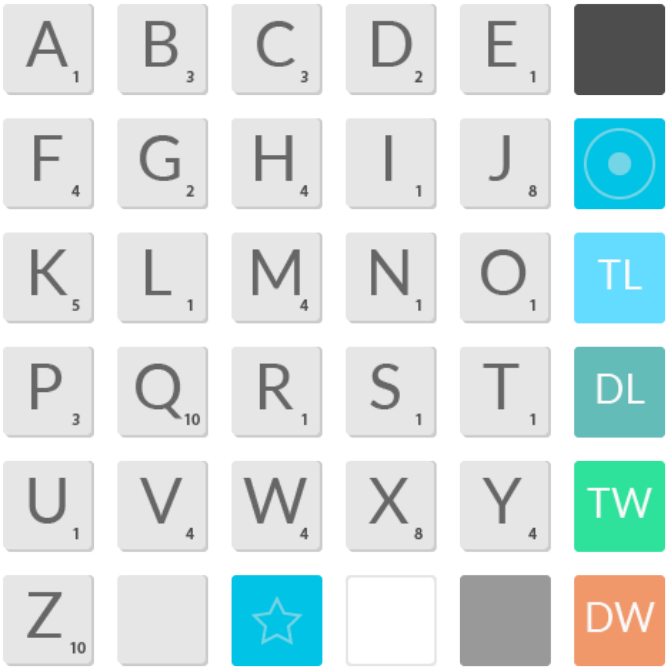
token.h

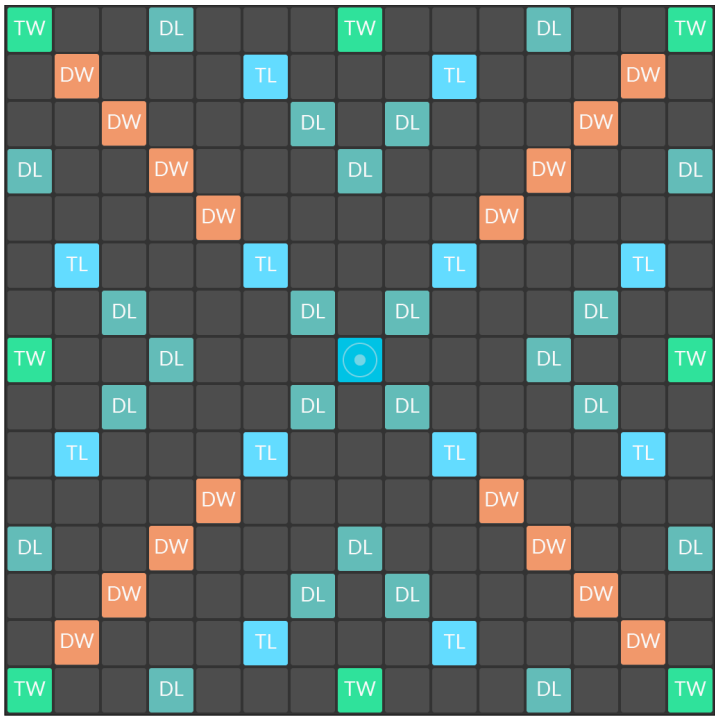
```
#ifdef TOKEN_H
#define TOKEN_H
#include <iostream>
#include <string>
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include "rectangle.h"

//Hereda del tipo rectangulo porque va contener una imagen
//y se encuentra en una coordenada (x,y),

class Token : public Rectangle
{
  private:
    //Valor de la letra
    int val;
    //Letra que contiene la imagen
    char key;

  public:

  Token(int, char);

  ~Token();

  inline int get_letter() const { return value; }

  inline int get_key() const { return key; }

};

#endif
```

main.cpp

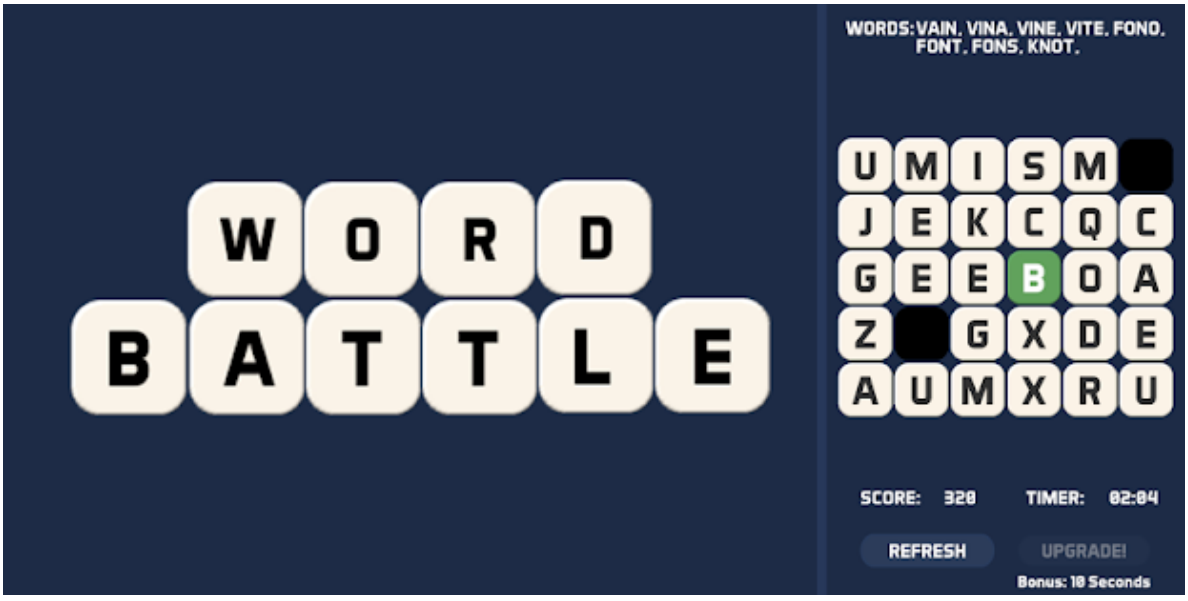
```
int main()
{
 //Se crea la ventana con un nombre y sus dimensiones
  Window window("Window test", 1300, 700);

  //Rectangulo que va a contener las imagenes
  Rect rect(window,180, 180, 0, 0, "/media/images/design.png");

  //Evento SDL. Este dato captura todos los dispositos I/O
  SDL_Event event;

  //Mientras que la ventana este abierta (no presiones la x en la esquina superior)
  while (!window.is_closed())
  {
    //Imprime los eventos del rectangulo por terminal
    rect.events();

    //Imprime los eventos de la ventana por terminal
    window.events();

    //Dibuja el rectangulo
    rect.draw();

    //Dibuja en la ventana
    window.clear();

  }
  return 0;
}
```

Para propósitos del avance y prueba de estructuras, trabajamos con las siguientes imágenes para representar el tablero y las fichas. Todos los diseños estan sujetos a cambios.





Prototipo de diseño final



En la imagen se aprecia el tablero (cortado para que se pudiese ver la terminal), y la terminal con los valores capturados por los eventos. Por ejemplo, entre las coordenadas (en pixeles) 720 < x < 780 20 < y < 100 se encuentra la letra A.