UNIVERSIDAD NACIONAL SAN AGUSTÍN DE AREQUIPA

FACULTAD DE INGENIERIA DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

Proyecto Final
Bejeweled Game
CHUQI INKA

Ciencias de la Computación 2
Integrantes:
Kemely Castillo Caccire
Profesor:
Alvaro Mamani Aliaga
Arequipa 2019

Índice

Índice de figuras

1. Descripción

Figura 1: Portada y nombre del proyecto

Los jugadores tienen que mover una joya para juntar tres o más joyas adyacentes del mismo color. Cuando esto ocurre, las joyas desaparecen y un nuevo grupo de gemas se generan aleatoriamente desde arriba para llenar el espacio vacío. A veces, se forman combinaciones automáticas, creando una cadena conocida como cascada. El juego termina cuando no hay mas movimientos posibles.

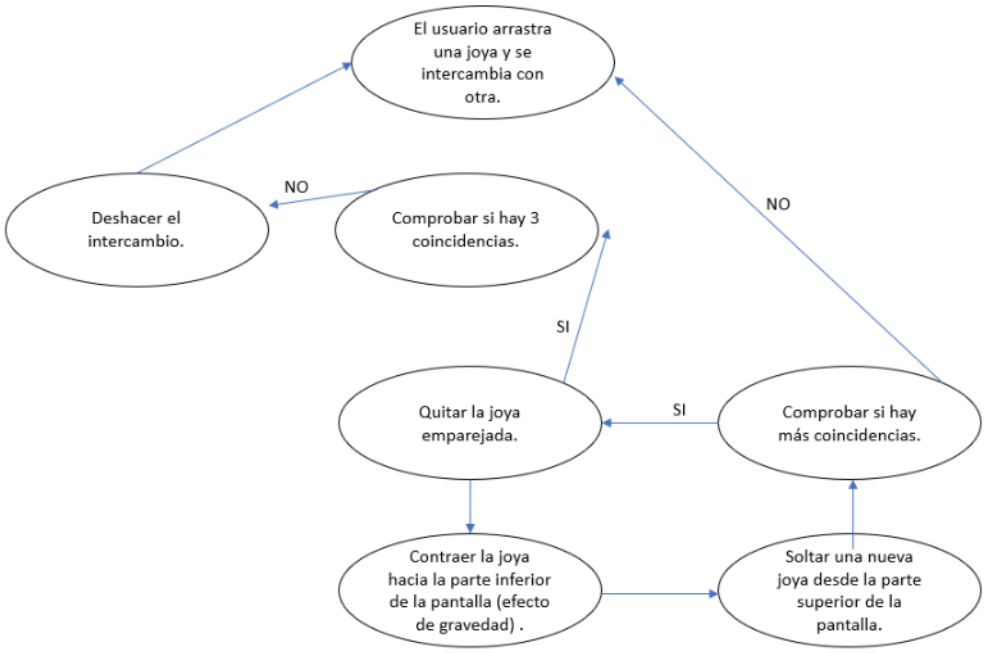

Figura 2: Diagrama de resumen del juego

- Lenguaje: C++ (Dev C++)
- Librerias: Simple and Fast Multimedia Library (SFML):System, Window, Graphics, Audio y Network.

2. Implementación

2.1. MAIN:

Creamos un objeto de clase MENU que nos dara una ventana,creada de acuerdo a los valores dados (int ancho,int alto,string titulo).

```
#include <iostream>
//incluimos la libreria grafica SFML
#include <SFML/Graphics.hpp>
#include "menu.h"
using namespace std;
int main()
{
  Menu partida;
  partida.iniciar(1200,800,"CHUQI INKA");
  partida.run();
  return 0;
}
```

2.2. CLASS MENU:

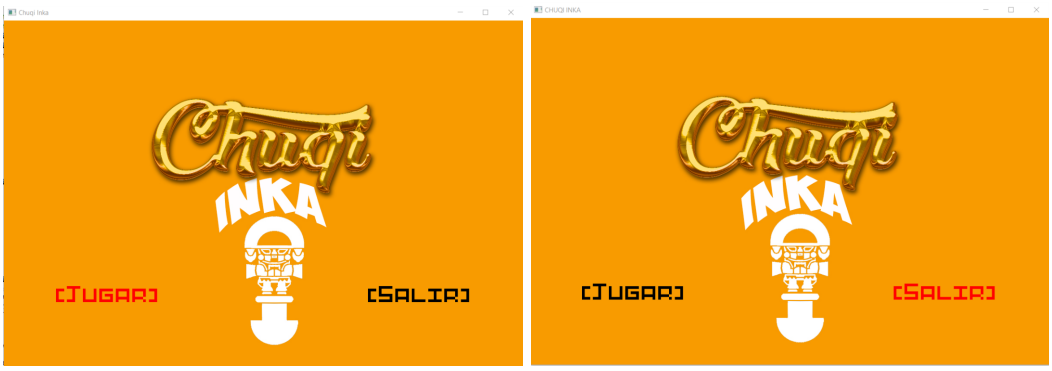

Figura 3: Menu.h y menu.cpp

La clase MENU nos va a generar una ventana con dos opciones (Jugar - Salir) la cual nos permitira acceder o salir a nuestro juego de manera didactica.

```
class Menu:public Sonido, public Ventana{
  private:
    RenderWindow ventana1;
    Font         fuente;
    Text         menu[NUMERO_DE_OPCIONES];
    Event        evento1;
    Texture      txt_fondo;
    Sprite       spr_fondo;
    int          select_item;
    Juego        juego;
  public:
    Menu() {}
    ~Menu() {}
    void iniciar(int ancho,int alto,string titulo);
    void cargar_graficos();
    void dibujar();
    void run();
};
```

- **void iniciar(int ancho,int alto,string titulo):**Crea la ventana ,limita el framerate a una frecuencia fija máxima .

```
void Menu::iniciar(int ancho,int alto,string titulo){
  ventana1.create(VideoMode(ancho,alto),titulo);
  ventana1.setFramerateLimit(60);
  cargar_graficos();
}
```

- **void cargar_graficos():**Carga los elementos de un archivo ,establece los colores,establecer el contenido del texto para nuestro menu.

```
void Menu::cargar_graficos(){
  if(!fuente.loadFromFile("fuentes/pixelart.ttf")){
    //Sale error
  }
  for(int i=0;i<NUMERO_DE_OPCIONES;i++){
    menu[i].setFont(fuente);
    menu[i].setCharacterSize(50);
  }
  menu[0].setFillColor(Color::Red);
  menu[1].setFillColor(Color::Black);
  menu[0].setPosition(120,600);
  menu[1].setPosition(840,600);
  menu[0].setString("[Jugar]");
  menu[1].setString("[Salir]");
    txt_fondo.loadFromFile("imagenes/portada.jpg");
  spr_fondo.setTexture(txt_fondo);
  Sonido::cargar_sonido();
  select_item=0;
}
```

- **void dibujar():**Nos va permitir mostrar en pantalla lo que se ha renderizado hasta en la ventana hasta el momento.

```
void Menu::dibujar(){
  ventana1.draw(spr_fondo);
  for(int i=0;i<NUMERO_DE_OPCIONES;i++){
    ventana1.draw(menu[i]);
  }
  ventana1.display();
}
```

- **void run():**Va a permitirnos iteractuar con el jugador ,por medio de los eventos(cerrar, teclado,click del mouse,movimiento del mouse).

```
void Menu::run(){
while(ventana1.isOpen()){
  while(ventana1.pollEvent(evento1)){
    switch(evento1.type){
        //Para cerrar la ventana
      case Event::Closed:
        ventana1.close();
        exit(1);
      break;
      //Utilizar las teclas
      case Event::KeyReleased:
        switch(evento1.key.code){
          case Keyboard::Left:
            sound_cam.play();
            if(select_item -1>=0){
                    menu[select_item].setFillColor(Color::Black);
                    select_item--;
                    menu[select_item].setFillColor(Color::Red);
                  }
          break;
          case Keyboard::Right:
            sound_cam.play();
            if(select_item +1<NUMERO_DE_OPCIONES){
                    menu[select_item].setFillColor(Color::Black);
                    select_item++;
                    menu[select_item].setFillColor(Color::Red);
                  }
          break;
          case Keyboard::Return:
            sound_select.play();
            switch(select_item){
            case 0:
               ventana1.close();
               juego.iniciar(1200,800,"JUEGO");
               juego.run();
               break;
            case 1:
               ventana1.close();
               break;
          break;
            }
          break;
```

```
        }
      }
      }
      //Llamar a la funcion Menu::Dibujar()
    dibujar();
   }
}
```

2.3. CLASS JUEGO:

Figura 4: Juego.h y juego.cpp

```
class Juego:public Sonido, public Ventana{
  private:
    RenderWindow ventana2;
      View          view_gema;
      View          view_pantalla;
      Texture       txt_juego;
      Sprite        spr_juego;
      Font          fuente;
      Text          text_puntaje;
      Text          text_final;
      Gestor        gestor_gema;
  public:
    Juego() :fuente(), text_puntaje("PUNTOS: 000000", fuente, 20),    text_final("Game Over!",
     fuente, 80)
      {
    text_puntaje.setFillColor(Color::Black);
    text_final.setFillColor(Color::Black);
      }
      ~Juego() {}
      void iniciar(int ancho,int alto,string titulo);
      void cargar_graficos();
      void dibujar();
      void run();

};
```

- **void iniciar(int ancho,int alto,string titulo):**Crea la ventana ,limita el framerate a una frecuencia fija máxima , nos ayuda a mantener la puntuación del texto en su lugar.Es importante ya que con esto vamos a cargar nuestro Sprite y la aleatoriedad(CLASE GESTOR).

```
void Juego::iniciar(int ancho,int alto,string titulo){
    ventana2.create(VideoMode(ancho,alto),titulo);
    ventana2.setFramerateLimit(60);
    cargar_graficos();
    // esto es solo para mantener la puntuacion del texto en su lugar.
    view_pantalla.reset(FloatRect(0, 0,ancho, alto));
    view_pantalla.setViewport(FloatRect(0.0f, 0.0f, 1.0f, 1.0f));
    double borde = (64.0f*8)+(4*7);
    view_gema.reset(FloatRect(0, 0, borde, borde));
```

```
    view_gema.setViewport(FloatRect(0.028125f, 0.05f, 0.50625f, 0.9f));
    ventana2.setView(view_gema);
    // Es importante ya que con esto vamos a cargar nuestro Sprite y la aleatoriedad(
    CLASE GESTOR)
    gestor_gema.iniciar();
}
```

- **void cargar_graficos():**Carga los elementos de un archivo (fuente,imagenes, sonido)

```
void Juego::cargar_graficos(){
    if (!fuente.loadFromFile("fuentes/aaa.ttf")) {
    }
    txt_juego.loadFromFile("imagenes/juego.jpg");
    spr_juego.setTexture(txt_juego);
    Sonido::cargar_sonido();
}
}
```

- **void dibujar():**Nos va permitir mostrar en pantalla lo que se ha renderizado hasta en la ventana hasta el momento , es importante el puntaje porque utilizaremos un tipo especial para imprimir el score.

```
void Juego::dibujar(){
    ventana2.setView(view_pantalla);
    ventana2.draw(spr_juego);
        //setw() -> tabulado
    ostringstream string_puntaje;
    string_puntaje << "PUNTOS: " << setw(6) << setfill('0') <<gestor_gema.getScore();
    text_puntaje.setString(string_puntaje.str());
    text_puntaje.setPosition(ventana2.getSize().x-450,120);
    text_final.setPosition(ventana2.getSize().x / 2, ventana2.getSize().y / 2);
        text_puntaje.setCharacterSize(40);
    ventana2.draw(text_puntaje);
    if (gestor_gema.pierde()) {
      ventana2.draw(text_final);
    }
    ventana2.setView(view_gema);
    gestor_gema.draw(ventana2);
    ventana2.display();
}
```

- **void run():**Va a permitirnos iteractuar con el jugador ,esta vez vamos a acceder a otras clases por medio de este evento.

```
void Juego::run()
  {
    while(ventana2.isOpen()) {
      Event e;
      while(ventana2.pollEvent(e)) {
        switch(e.type) {
          case Event::Closed:
            ventana2.close();
            break;
          case Event::KeyReleased:
            if (e.key.code == Keyboard::Escape) ventana2.close();
            if (e.key.code == Keyboard::W) gestor_gema.generar_premio();
            sound_glow.play();
            break;
          case Event::MouseButtonPressed:
            sound_cam.play();
            if (e.mouseButton.button == Mouse::Left) {
              gestor_gema.click(ventana2.mapPixelToCoords(Mouse::getPosition(ventana2)));
            }
            break;
          default:
            break;
        }
      }

      gestor_gema.update();
      dibujar();
  }
}
```

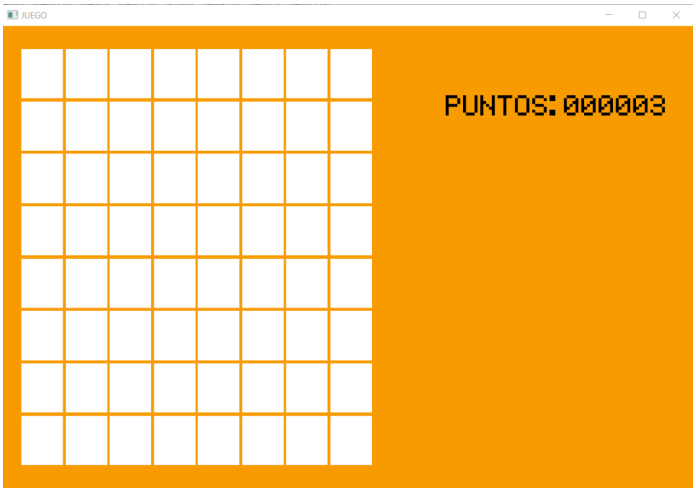

Figura 5: Gema.h y gema.cpp

2.4. CLASS GEMA:

La clase gema , nos va a permitir actualizar los estados o color de las gemas al igual que asignarles el espacio en que van ha ser mostradas.

```
class Gema{
  public:
  //Mejora la legitividad del codigo asignandole valores de 0,...,n(cantidad de variables del
    enum)
    enum class Color : int
    {
      ROJO , NARANJA , AMARILLO , VERDE , AZUL , MORADO , BLANCO , PREMIO
  };
  enum class Estado : int
  {
    NUEVO, NINGUNO, SELECCIONAR, IGUAL, MOVER, BORRAR, ELIMINAR
  };
    Gema(int col, int fila, Color color, Texture& txt_gema, Estado estado);
      ~Gema() = default;
    void dibujar(RenderWindow& window);
      bool comprobar(const Vector2f& spos);
      void intercambiar_obj(Gema& otra);
      Estado actualizar();

      int getCol() const { return col; }
      int getFila() const { return fila; }
      void setColor(Color color) { this->color = color; }
      Color getColor() const { return color; }
      void setEstado(Estado estado);
      Estado getEstado() const { return estado; }

      //conservara su valor hasta la ejecucion del programa y, ademas, no aceptara ningun
    cambio en su valor
      static const Vector2f size;
      static constexpr int padding = 4;
      static constexpr int gemSize = 64;
  private:
    int          col, fila;
      Vector2f     pos;
      Vector2f     objetivo;
      Sprite       spr_gema;
      Texture      txt_gema;
      Color        color;
      Estado       estado;
      int          alpha = 255;
};
```

- **Gema(int col, int fila, Color color, Texture& txt_gema, Estado estado):**constructor de la clase GEMA donde iniciaremos varios valores que luego seran uritilizados en la clase GESTOR para crear objetos de GEMA, nos genera un sub-rectángulo de la textura que mostrará el sprite ,construye el rectángulo a partir de sus coordenadas y obtener la posición del objeto.

```
const Vector2f Gema::size {(float)gemSize, (float)gemSize};
Gema::Gema(int col, int fila, Color color, Texture& txt_gema, Estado estado)
  : col(col), fila(fila), pos(float((Gema::size.x+padding)*col), float((Gema::size.y+
    padding)*fila)),color(color), estado(estado), spr_gema(txt_gema)
{
  objetivo = pos;
  spr_gema.setTextureRect(IntRect(static_cast<int>(color)*gemSize, 0, gemSize, gemSize));
  spr_gema.setColor(sf::Color(255, 255, 255, alpha));
  spr_gema.setPosition(pos);
}
```

Figura 6: Sprite en el constuctor GEMA

- **void dibujar(RenderWindow& window):**Recira como paramemtro una ventana ,el sprite generara un rectangulo de acuerdo al valor del Color y el gemSize definido(64)ademas cuando se eliminan las gemas disminuyen su opacidad lo cual representa el alpha para generar un Efecto de DESVANECER.

```
void Gema::dibujar(RenderWindow& window)
{
  if (estado != Gema::Estado::NUEVO) {

    spr_gema.setPosition(pos);
        spr_gema.setTextureRect(IntRect(static_cast<int>(color)*gemSize,    (estado==
    Estado::SELECCIONAR)?gemSize:0, gemSize, gemSize));
        spr_gema.setColor(sf::Color(255, 255, 255, alpha));
    window.draw(spr_gema);

  }
}
```

- **bool comprobar(const Vector2f& spos):**Comprueba si un punto está dentro del área del rectángulo,si el punto se encuentra en el borde del rectángulo, esta función devolverá el valor falso.

```
bool Gema::comprobar(const Vector2f& spos)
{
  if (estado==Estado::ELIMINAR) return false;
  return Rect<float>(pos, size).contains(spos);

}
```

- **void intercambiar_obj(Gema& otra):**Intercambia los contenidos de este buffer de vértice con los de otro ya que el objetivo==posicion (tambien se modifican las posiciones),el estado de las dos gemas se vuelve a MOVER.

```
void Gema::intercambiar_obj(Gema& otra)
{
  swap(col, otra.col);
  swap(fila, otra.fila);
  objetivo = Vector2f(float((Gema::size.x+padding)*col), float((Gema::size.y+padding)*
    fila));
  otra.objetivo = Vector2f(float((Gema::size.x+padding)*otra.col), float((Gema::size.y+
    padding)*otra.fila));
  setEstado(Estado::MOVER);
  otra.setEstado(Estado::MOVER);
}
```

- **Estado actualizar():**Es importante ya que va ha tomar valores de la clase enum Estado,y segun este va ha realizar diferentes acciones (revisar la clase GESTOR).

```
Gema::Estado Gema::actualizar()
{
  switch(estado)
  {
    case Estado::IGUAL:
    {
      setEstado(Estado::BORRAR);
      break;
    }
    case Estado::BORRAR:
    {
        //disminuye la opacidad(alpha)
      alpha -= 10;
      if (alpha <= 50) {
        setEstado(Estado::ELIMINAR);
      }
      break;
    }
    case Estado::MOVER:
    case Estado::ELIMINAR:
    {
      auto move = [=](float& p, float t) -> bool
      {
        // mostrar la apariencia del movimiento
        if (p == t) return false;
        if (p < t) p += min(5.0f, t-p);
        else       p -= min(5.0f, p-t);
        return true;
      };
      if (!move(pos.x,objetivo.x) && !move(pos.y, objetivo.y) && estado == Estado::MOVER)
        setEstado(Estado::NINGUNO);
      break;
    }
    default:
      break;
  }
  return estado;
}
```

- **void setEstado(Estado estado):**Se le asigna el estado

```
void Gema::setEstado(Estado estado)
{
  if (this->estado != Estado::ELIMINAR) this->estado = estado;
}
```

2.5. CLASS GESTOR:

Figura 7: Gestor.h y gestor.cpp

```
class Gestor{
public:
  Gestor() = default;
  ~Gestor() = default;

  bool iniciar();
  void click(const Vector2f& spos);
  void cargar_graficos();
  void reiniciar();
  void update();
  void draw(RenderWindow& window);
  bool generar_premio();
  bool pierde();

  static constexpr int filas = 8;
  static constexpr int cols = 8;
  static constexpr int puntosPremio = 10;

  int getScore() const { return score; }

private:
  enum class State : int
  {
    WAITING, SELECTED, SWAPPING, MOVING, ARRANGING, LOSING
  };

  void setState(State newState);
  int match();
  int match3(Gema* gem1, Gema* gem2, Gema* gem3);
  bool organizar();
  void reemplazar_eliminado();
  void explotar(Gema* gem);

  vector<Gema> gems;
  Texture      texture;
  int          sel1;
  int          sel2;
  State        state = State::WAITING;
  int          score = 0;
  int          cantidad_premio;
};
```

- **iniciar():** Es imprtante ya que con esto empezamos la aleatoriedad de nuestras gemas y les otorgamos el ESTADO::NUEVO(0) es decir reiniciamos todas las gemas.

```
bool Gestor::iniciar(){
  cargar_graficos();
  srand(time(0));
  reiniciar();
  return true;
}
```

- **cargar_graficos():**Carga los elementos de un archivo(en esre caso nuestro sprite de gemas)

```
void Gestor::cargar_graficos(){
  texture.loadFromFile("imagenes/gemas.png");
}
```

- **reiniciar():** VAMOS A tener un vector de objetos de la class GEMA llamado vector ¡Gema¿gems
  Elimina todos los elementos del vector,eL ESTADO DE todas las gemas es NINGUNO(0) .Se utiliza para insertar un nuevo elemento en el contenedor de vectores, el nuevo elemento se agrega al final del vector emplace_back a diferencia de push_back permite construir sobre la marcha, por lo tanto, solo se llama constructor, no hay movimiento,esta funcion pertenece al la libreria ¡algorithm¿como ven inserta los valores en orden deacuerso a nuestro constuctor de la clase Gema(int col, int fila, Color color, Texture& txt_gema, Estado estado).El ESTATE de todas las gemas es WAITING (0).

```
void Gestor::reiniciar(){
  gems.clear();
  for (int r = 0; r < filas; ++r) {
    for (int c = 0; c < cols; ++c) {
      gems.emplace_back(c, r, static_cast<Gema::Color>(rand()%7), texture, Gema::Estado::
    NINGUNO);
    }
  }
```

```
  score = 0;
  cantidad_premio = 0;
  //el ESTATE DE todas las gemas es WAITING (0)
  setState(State::WAITING);
}
```

- **draw(RenderWindow& window):**Dibujamos las gemas ,vamos a la clase Gema::dibujar.

```
void Gestor::draw(RenderWindow& window)
{
  for (auto& g : gems) {
    g.dibujar(window);
  }
}
```

- **click(const Vector2f& spos):**

```
void Gestor::click(const Vector2f& spos)
{
  if (state != State::WAITING && state != State::SELECTED) return;
  for (auto& gem : gems) {
    // bool Gema::comprobar(const Vector2f& spos)
    if (gem.comprobar(spos)) {
      if (gem.getColor() == Gema::Color::PREMIO) {
        // premio explota
        return explotar(&gem);
      }
      if (state == State::SELECTED) {
        sel2 = gem.getFila() * cols + gem.getCol();
        auto& other = gems[sel1];
        if ((abs(gem.getCol()-other.getCol()) + abs(gem.getFila()-other.getFila())) != 1)
     break;
        iter_swap(&gem, &other);
        other.intercambiar_obj(gem);
        setState(State::SWAPPING);
      } else {
        sel1 = gem.getFila() * cols + gem.getCol();
        gem.setEstado(Gema::Estado::SELECCIONAR);
        setState(State::SELECTED);
      }
      break;
    }
  }
}
```

- **update():**

```
void Gestor::update()
{
  if (state == State::WAITING) {
    for (auto& gem : gems) {
      if (gem.getEstado() == Gema::Estado::NUEVO) gem.setEstado(Gema::Estado::NINGUNO);
    }

    int m = match();
    if (m > 0) {
      setState(State::MOVING);
      score += m;
    } else if (organizar()) {
      setState(State::MOVING);
    }

  }
  if (state == State::MOVING || state == State::SWAPPING) {
    bool moving = false;
    for (auto& gem : gems) {
      Gema::Estado estado = gem.actualizar();
      if (estado == Gema::Estado::MOVER || estado == Gema::Estado::BORRAR) moving = true;
    }
    if (!moving) {
      if (state == State::SWAPPING) {
        int m = match();
        score += m;
        if (m == 0) {
          auto& gem = gems[sel1];
          auto& otra = gems[sel2];
```

```
          iter_swap(&gem, &otra);
          otra.intercambiar_obj(gem);
        }
        setState(State::MOVING);
      } else {
        setState(State::WAITING);
      }
    }
  }

  if (state == State::LOSING) {
    for (auto& gem : gems) {
      gem.setEstado(Gema::Estado::BORRAR);
      gem.actualizar();

    }
  }
}
```

- **match():**

```
int Gestor::match()
{
  int match = 0;
  auto gem = gems.begin();
  for (int r = 0; r < filas; ++r) {
    for (int c = 0; c < cols; ++c, ++gem) {
      if (c > 0 && c < (cols-1)) {
        match += match3(&*gem , &*(gem+1), &*(gem-1));
      }
      if (r > 0 && r < filas-1) {
        match += match3(&*gem , &*(gem+cols), &*(gem-cols));
      }
    }
  }
  return match;
}
```

- **match3(Gema* gem1, Gema* gem2, Gema* gem3):**

```
int Gestor::match3(Gema* gem1, Gema* gem2, Gema* gem3)
{
  if (gem1->getEstado() != Gema::Estado::NINGUNO && gem1->getEstado() != Gema::Estado::
    IGUAL
      && gem2->getEstado() != Gema::Estado::NINGUNO && gem2->getEstado() != Gema::Estado
    ::IGUAL
      && gem3->getEstado() != Gema::Estado::NINGUNO && gem3->getEstado() != Gema::Estado
    ::IGUAL) {
    return 0;
  }
  int match = 0;
  if (gem1->getColor() == gem2->getColor() && gem1->getColor() == gem3->getColor()) {
    if (gem1->getEstado() != Gema::Estado::IGUAL) {
      ++match;
      gem1->setEstado(Gema::Estado::IGUAL);
    }
    if (gem2->getEstado() != Gema::Estado::IGUAL) {
      ++match;
      gem2->setEstado(Gema::Estado::IGUAL);
    }
    if (gem3->getEstado() != Gema::Estado::IGUAL) {
      ++match;
      gem3->setEstado(Gema::Estado::IGUAL);
    }
  }
  return match;
}
```

- **organizar():**Utilizamos el ITERATOR que devuelve un iterador inverso que apunta al último elemento del vector, con la que vamos a recorrer el vector de clase GEMA.

```
bool Gestor::organizar()
{
  bool organizado = false;
  for(auto gem = gems.rbegin(); gem != gems.rend(); ++gem) {
    if (gem->getEstado() != Gema::Estado::ELIMINAR) continue;
```

```
    if (gem->getFila() == 0) break;
    organizado = true;
    for (int fila = 1; fila <= gem->getFila(); ++fila) {
      if ((gem+(cols*fila))->getEstado() != Gema::Estado::ELIMINAR) {
        iter_swap(gem, (gem+(cols*fila)));
        gem->intercambiar_obj(*(gem+(cols*fila)));
        break;
      }
    }
  }
  reemplazar_eliminado();
  return organizado;
}
```

- **reemplazar_eliminado()**

```
void Gestor::reemplazar_eliminado()
{
  for(auto gem = gems.begin(); gem != gems.end(); ++gem) {
    if (gem->getEstado() == Gema::Estado::ELIMINAR) {
      *gem = Gema(gem->getCol(), gem->getFila(), static_cast<Gema::Color>(rand()%7),
    texture, Gema::Estado::NUEVO);
    }
  }
}
```

- **setState(State newState):**Asignamos un Estate

```
void Gestor::setState(State newState)
{
  state = newState;
}
```

- **Gestor::explotar(Gema* gem)**

```
void Gestor::explotar(Gema* gem)
{
  for(int r = gem->getFila()-1; r < gem->getFila()+2; ++r) {
    for(int c = gem->getCol()-1; c < gem->getCol()+2; ++c) {
      if (c >= 0 && c < cols && r >=0 && r < filas) {
        gems[r * cols + c].setEstado(Gema::Estado::IGUAL);
      }
    }
  }
  setState(State::MOVING);

}
```

- **generar_premio()**

```
bool Gestor::generar_premio()
{
  if (score >= puntosPremio) {
    cantidad_premio++;
    gems[(rand()%filas)*cols + (rand()%cols)].setColor(Gema::Color::PREMIO);
    score -=puntosPremio;

    return true;
  }

  return false;
}
```

- **pierde()**

```
bool Gestor::pierde()
{
  if (state == State::LOSING)
    return true;
  else
    return false;
}
```

2.6. CLASS AUDIO:

La clase AUDIO nos va a ayudar con el manejo del sonido , va a HEREDAR los atributos a la clases que neseciten audio , y Polimorfismo en la función virtual void cargar_sonido().

```
class Sonido{
  public:
    SoundBuffer  bff_cam;
    Sound        sound_cam;
    SoundBuffer  bff_select;
    Sound        sound_select;
    SoundBuffer  bff_glow;
    Sound        sound_glow;
    virtual void cargar_sonido();
  Sonido() = default;
  ~Sonido() = default;
};
```

- **virtual void cargar_sonido():** Cargamos el buffer de sonido desde un archivo (CARPETA sonido) y adjuntamos al sonido.

```
void Sonido::cargar_sonido(){
  bff_cam.loadFromFile("sonido/cambio.wav");
  sound_cam.setBuffer(bff_cam);
  bff_select.loadFromFile("sonido/select.wav");
  sound_select.setBuffer(bff_select);
  bff_glow.loadFromFile("sonido/glow.wav");
  sound_glow.setBuffer(bff_glow);
}
```

2.7. CLASS VENTANA:

Para utiliza Polimorfismo ,implemente esta clase padre VENTANA que se usa para representar aquellos métodos que después se implementarán en las clases derivadas MENU y JUEGO(ya que contiene funciones virtuales puras).

```

//clase abstracta -> polimorfismo
class Ventana{
  public:
      virtual void iniciar(int ancho,int alto,string titulo)=0;
    virtual void cargar_graficos()=0;
    virtual void dibujar()=0;
    virtual void run()=0;

};
```

3. Problematica

3.1. Problemas encontrados

- El uso de la libreria SFML , al ser la primera vez en trabajarla :
  - El uso de texturas.
  - El uso de sprites.
  - Como debe iteractuar , el uso de eventos.
  - Al ser una libreria 2D(x,y), el uso de vectores de este tipo.
- El uso de clases , para que iteractuen entre ellas.

3.2. Soluciones

- En cuanto a la libreria , el uso de tutoriales o la pagina oficial de SFML.
- La consulta ha proyectos en esta misma libreria.

4. Conclusiones

- El uso de la herencia y polimorfismo evita la redundancia de atributos y funciones ,que usualmente tienen nombres o acciones parecidas.
- El uso del metodo de diseño ITERATOR(Vector) es importante ya que podemos recorrer objetos que nosotros creemos de manera eficaz.
- El uso de enum class nos hace ver el codigo de una manera mas entendible .
- La libreria The Standard Template Library (STL) (vector y algoritmo)nos ayudan al tener las funciones que deseamos de manera ya definida.

5. Cronograma

Actividad	MES											
	MAYO				JUNIO				JULIO			
	SEMANAS											
	1era	2da	3era	4ta	1era	2da	3era	4ta	1era	2da	3era	4ta
Pensar ideas(proyecto)												
Buscar informacion												
Tutoriales de la librería SFML												
Clase MENU y JUEGO												
Creación (textura menu)												
Presentación(primer avance)												
Clase GEMA												
Crear el Sprite (para las gemas)												
Clase GESTOR												
Implementar la herencia y polimorfismo(Clase Sonido)												
Iterador												
Exposición(trabajo)												
Clase abstracta(Ventana)												
Elaboración del informe												

Figura 8: Cronograma de actividades proyecto

6. Diagrama de clases

Figura 9: Diagrama de clases

7. Link

- Repositorio del proyecto:
  https://github.com/kemely2018/CHUQI-INKA