Programación
1º DAM

Proyecto Final

ONEIRIC

I.E.S. San Vicente
San Vicente del Raspeig (Alicante)
Curso 2018/2019

Alumnos:
Kevin Marín
Jaime Rebollo

Profesor:
Nacho Cabanes

1. Introducción

Nombre del proyecto
Oneiric

Desarrollado por
Kevin Marín
Jaime Rebollo

Descripción breve del proyecto
Es un juego estilo RPG con batallas dinámicas estilo Pokemon (1 vs 1), en el cual podrás manejar al protagonista X debiendo recorrer la senda del programador. Tendrás que avanzar a lo largo del mapa e ir consiguiendo fragementos de Código para conseguir habilidades nuevas y enfrentarte a los grandes Boses. Estará desarrollado con C# usando la biblioteca Tao.SDL.

2. Funcionalidad del proyecto

Cuando inicias el juego te aparece el logo de la empresa que lo ha desarrollado, después llegarás al menú principal, en el cual podrás elegir entre:

- **Continuar**: Donde se cargará la última jugada.
- **Nueva Partida**: Donde se empezará el juego desde cero.
- **Cargar Partida**: Mostrará los datos de la partida, donde el juagdor podrá elegir entre un máximo de 3 partidas guardadas.
- **Opciones**: Donde el jugador podrá cambiar las opciones del juego, entre ellas el idioma (Español-Ingles), también elegir el nivel de dificultad (Fácil-Medio-Dificil-Hacker). Los niveles de dificultad se diferenciarán entre ellos únicamente por un multiplicador que se aplicará a todos los daños que hagan los enemigos. También cambiar a pantalla completa y elegir el nivel de volumen.
- **Ayuda**: Donde el jugador podrá consultar los controles del juego.
- **Salir**: Se podrá salir del juego.

En cualquiera de las 3 primeras opciones comenzará una partida, donde aparecerá el mapa y el personaje. Según donde el jugador se mueva hacia los extremos del mapa, éste se irá moviendo entre fragmentos del mapa. A lo largo del mapa existirán zonas en las cuales cada X pasos (número aleatorio, reseteandose después de cada lucha) aparecerá un combate, en el caso de vencer tú experiencia aumentará y podrás conseguir frágmentos de código de distintos tipos, mientrás que si pierdes resucitarás en el último punto guardado.

Al conseguir x cantidad de frágmentos podrás crear una sentencia de código completa, según que sentencia recibiras una recompensa u otra.

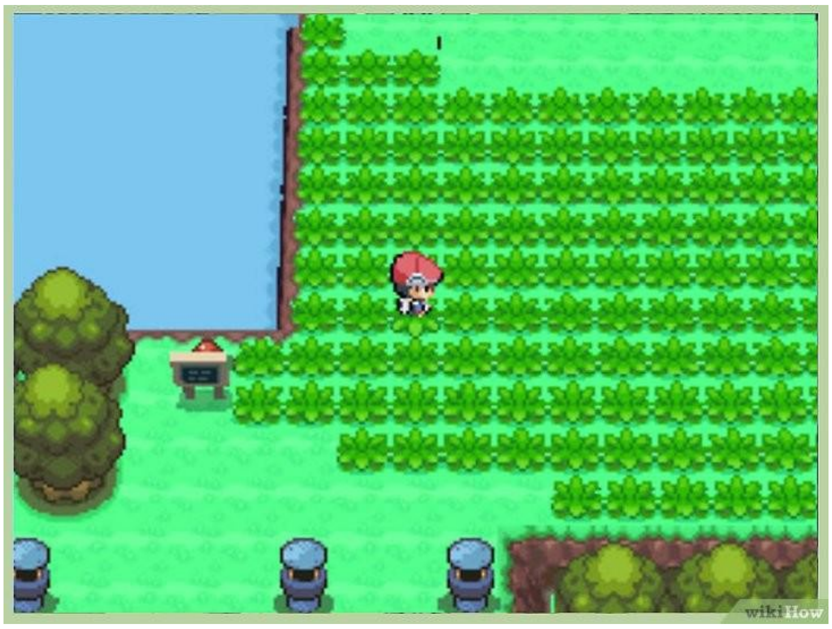
El mapa estará ambientado en un instituto exageradamente grande, en algunas zonas podrás encontrar ordenadores, los cuales te podrán dar x cantidad de fragmentos u otros objetos.

El jugador también tendrá un inventario con objetos que podrá utilizar durante el combate o fuera de él, además de equipables.

Habrán zonas en las que encontraremos boses, para poder enfrentarnos correctamente a ellos necesitaremos poseer un cierto nivel y unas específicas habilidades.

3. Prototipo de la pantalla

La apariencia que se persigue es ésta:



4. Entregas previstas

1. ~~Hacer el esqueleto del juego, incluyendo todas las clases necesarias~~
2. ~~Realizar el menú principal del juego, incluyendo todos los submenús~~
3. ~~Movimiento del personaje y combates aleatorios~~
4. ~~Cargar y dibujar mapa desde archivos y colisiones y scroll~~
5. ~~Guardado de partida e interacción con elementos estáticos (cofres/ordenadores)~~
6. ~~Implementación del menú in-game (Al pulsar I)~~
7. ~~Creación de la interfaz de combate y los enemigos~~
8. ~~Realizar lógica de los combates~~
9. ~~Implementar lógica de los enemigos (normales)~~
10. Creación de Eventos y NPCs (no enemigos)
11. Implementación de habilidades del personaje
12. Agregar Bosses
13. Añadir música y sonidos
14. Realizar zonas de extras
15. Implementar sistema de trucos avanzado (inmortalidad, aumento de estadísticas)
16. Añadir Modo Supervivencia (muchos combates seguidos sin descanso)

5. Trabajo diario realizado

- 2019/04/17 – Se han creado el esqueleto del juego incluyendo todas sus clases, además se ha organizado todo en subcarpetas.
- 2019/05/06 – Creación y organización del menú principal y el submenú de opciones.
- 2019/05/13 – Se ha implementado el movimiento del personaje, cargar y dibujar el mapa desde un archivo, colisiones y scroll.

- 2019/05/15 – Implementación de multilenguaje, creación del menú *in-game* y creación de temporizador para tiempo jugado.
- 2019/05/16 – Se ha implementado el guardado y cargado de partidas además de crear fichero donde se guardarán los errores que ocurran. Para complementar el guardado se ha implementado la interfaz básica del guardado en el menu in-Game.
- 2019/05/17 – Se ha corregido el sistema de guardado y se ha implementado el guardado de las opciones y su persistencia al cerrar el juego. Se ha completado la clase Chest.
- 2019/05/20 – Implementación del inventario del personaje, interacción con los cofres y asignación aleatoria de *items* a los cofres.
- 2019/05/22 – Se han añadido los atributos restantes al jugador, se han creado las clases de los enemigos y se ha actualizado el guardado y carga de ficheros con los datos correspondientes.
- 2019/05/23 – Se ha mejorado el sistema de combate.
- 2019/05/24 – Implementada la lógica de combate y el historial de combate.
- 2019/05/27 – Mejora del historial de combate, implementación del drop de objetos al derrotar enemigos.
- 2019/05/29 – Listado de objetos en menu juego, implementado uso de objetos.
- 2019/05/30 – Mejorado el uso de objetos, posibilidad de usar objetos consumibles en batalla. Finalización del juego al morir el personaje principal enviándolo a GameOver y mostrar estadísticas del jugador en el menú *in-game*.
- 2019/05/31 – Diseño del primer mapa actualizado y mostrar los iconos para las estadísticas del jugador.

6. Problemas encontrados durante el desarrollo y sus soluciones

- 2019/04/17 – Tras crear las clases hemos necesitado organizarlas en directorios. Para ello hemos creado las carpetas, hemos movido los fuentes a sus respectivos directorios y hemos modificado las rutas de los mismos en el fichero ".csproj".
- 2019/04/27 – Se ha encontrado el problema de que al matar el enemigo y que éste te de un objeto, el programa da un NullPointerException.
  El problema resultó ser que no cargaba el fichero de los objetos.

7. Estructuras utilizadas

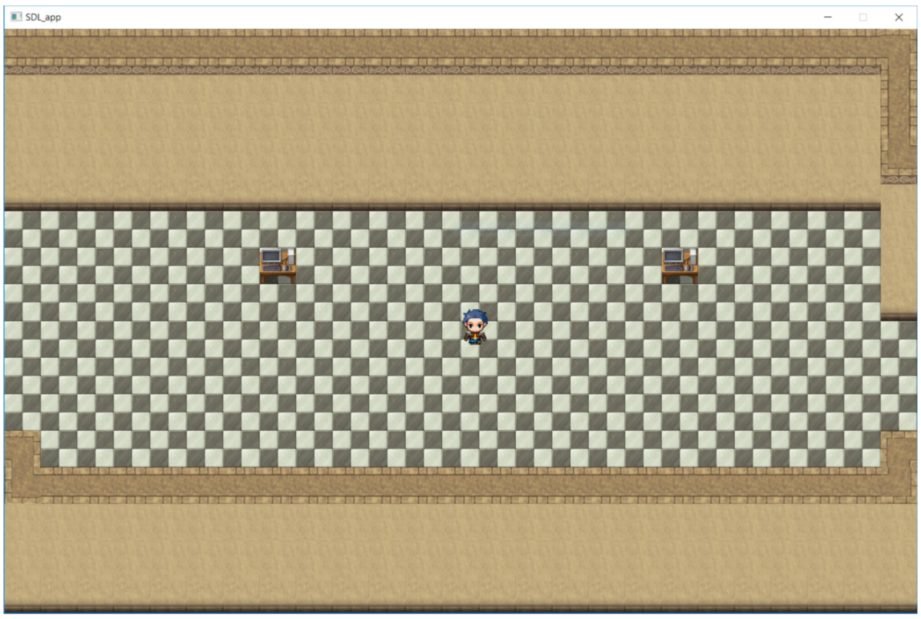
- ~~if~~
- ~~else~~
- ~~Conectores: && y/o || y/o !~~
- ~~switch~~
- ~~?~~
- ~~while~~
- ~~for~~
- ~~foreach~~
- ~~try-catch~~
- ~~(arrays)~~
- ~~struct~~
- ~~(clases + herencia)~~
- ~~(propiedades o getters y setters)~~
- ~~public, protected, (opcional) private~~

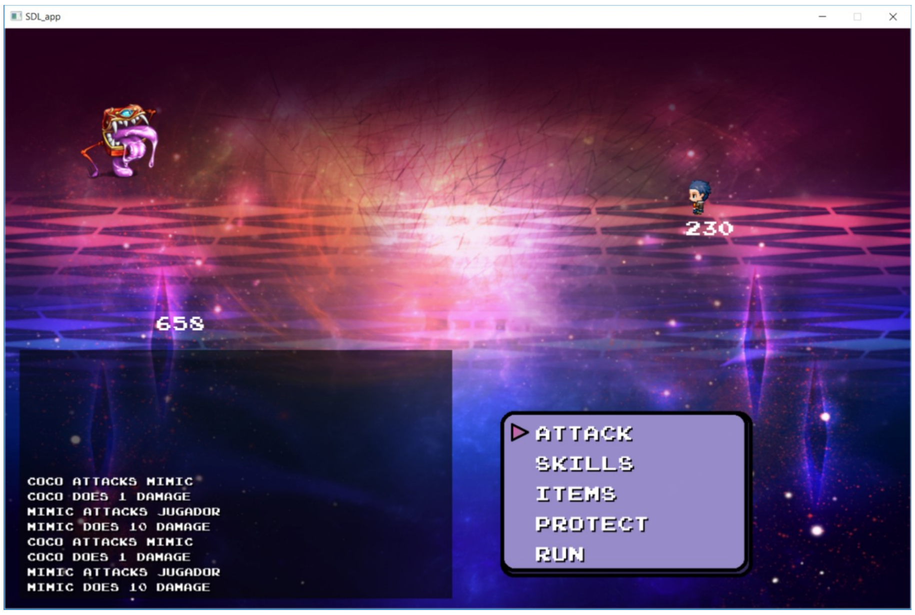
- ~~ArrayList o List<>~~
- ~~Hashtable o SortedList o Dictionary~~
- ~~StreamReader o FileStream o BinaryReader~~
- ~~ref o out~~
- ~~(manejo avanzado de cadenas: substring, contains, split, replace o similares)~~
- ~~(consola avanzada o SDL o Windows Forms o Unity)~~

8. Mejoras o restricciones respecto a la idea inicial

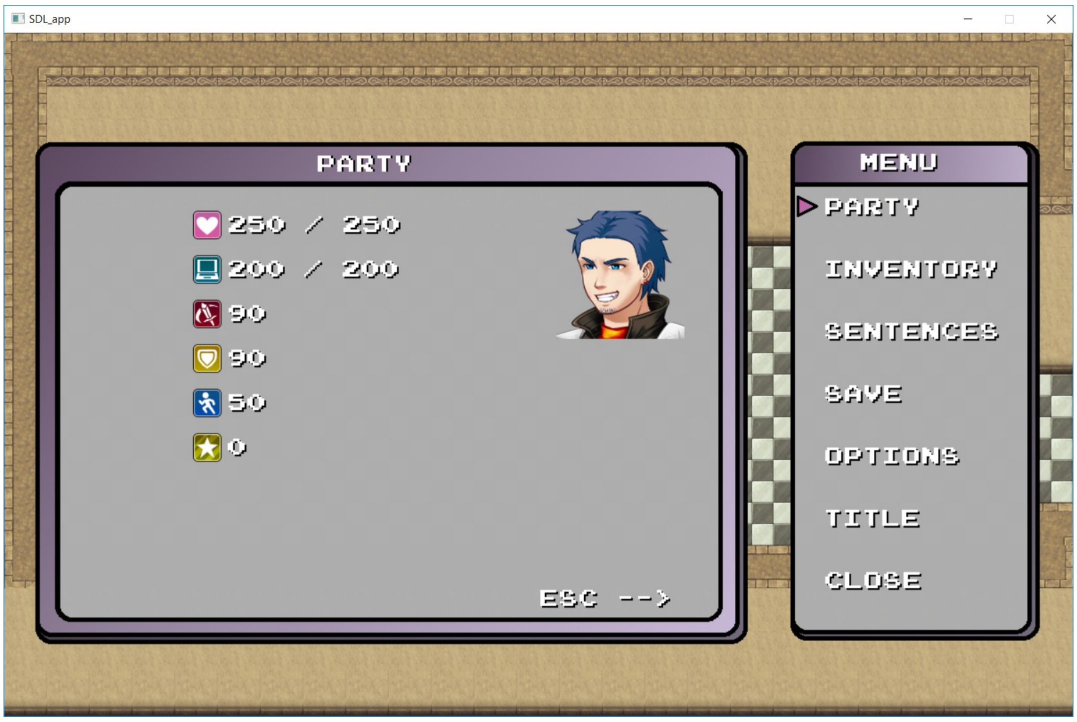
- Debido a la falta de tiempo faltan bastantes partes del juego, como armas, habilidades, jefes…
- Debido al uso de SDL no se puede jugar en pantalla completa, porque se ralentiza mucho.

9. Capturas de pantalla del proyecto final





```
COCO ATTACKS MIMIC
COCO DOES 1 DAMAGE
MIMIC ATTACKS JUGADOR
MIMIC DOES 10 DAMAGE
COCO ATTACKS MIMIC
COCO DOES 1 DAMAGE
MIMIC ATTACKS JUGADOR
MIMIC DOES 10 DAMAGE
```



10. Código fuente del proyecto final

```
using System;
using System.Collections.Generic;

class BattleScreen : Screen
{
    protected Image selector, menu, player, historyI, selector2, items;
    protected static int option;
    protected Font font72, font16;

    const int YCURSOR_MAX = 4;
    const int YCURSOR_MIN = 0;
    protected static NormalEnemy enemy;
    protected Queue<string> history;
    protected Dictionary<string,string> battleTexts;
    protected bool protecting;

    public BattleScreen()
        : base(new Image("data/images/other/battleBackground.png"),
            new Font("data/fonts/Joystix.ttf", 28))
    {
        option = 0;
        selector = new Image("data/images/other/selector2.png");
        selector2 = new Image("data/images/other/selector2.png");
        menu = new Image("data/images/other/screen_battle.png");
        items = new Image("data/images/other/screen_battle.png");
        historyI = new Image("data/images/other/history.png");
        player = new Image("data/images/player/Left_2.png");
        font72 = new Font("data/fonts/Joystix.ttf", 72);
        font16 = new Font("data/fonts/Joystix.ttf", 16);
        texts = new Dictionary<string, string>();
        battleTexts = new Dictionary<string, string>();
        history = new Queue<string>();
        for (int i = 0; i < 15; i++)
        {
            history.Enqueue(" ");
        }
        protecting = false;
    }

    public int GetChosenOption()
    {
        return option;
    }

    public void Run()
    {
        bool endBattle = false;
        LoadText(Oneiric.Languages[Oneiric.Language], "battleTexts");

        SdlHardware.Pause(100);
        PrepareBattle();
        do
        {
            bool endPlayerTurn = false;
            do
            {
                PlayerTurn(ref endBattle, ref endPlayerTurn);
                UpdateScreen();
                SdlHardware.Pause(100);
            } while (!endPlayerTurn);

            SdlHardware.Pause(500);
            if (!endBattle)
            {
                EnemyTurn(ref endBattle);
                UpdateScreen();
                SdlHardware.Pause(100);
            }
        }
        while (!endBattle);

    }

    public void UpdateScreen() {
        SdlHardware.ClearScreen();
        DrawMenu();
        ShowHistory();
        SdlHardware.ShowHiddenScreen();
    }

    public void PlayerTurn(ref bool endBattle, ref bool endPlayerTurn)
    {
        if (protecting)
        {
            Oneiric.g.Mcharacter.Defense /= 2;
        }
        if (SdlHardware.KeyPressed(SdlHardware.KEY_W) && option >
                YCURSOR_MIN)
        {
            option--;
        }
        else if (SdlHardware.KeyPressed(SdlHardware.KEY_S) && option <
            YCURSOR_MAX)
        {
            option++;
        }
        else if (SdlHardware.KeyPressed(SdlHardware.KEY_ESC))
        {
            option = YCURSOR_MAX;
        }
        else if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
        {
            SelectedOption(ref endBattle, ref endPlayerTurn);
        }
    }

    public void EnemyTurn(ref bool endBattle)
    {
        WriteOnHistory(enemy.GetType() + texts["aa"] +
                    " Jugador");
        string damage = enemy.Attack(Oneiric.g.Mcharacter);
        WriteOnHistory(enemy.GetType() + texts["in"] +
                    " " + damage +  texts["dm"]);
        if (Oneiric.g.Mcharacter.ActualLife == 0)
        {
            WriteOnHistory(texts["yd"]);
            endBattle = true;
        }
    }

    public void SelectedOption(ref bool endBattle, ref bool endPlayerTurn)
    {
        switch (option)
        {
            case 0:
                WriteOnHistory(Oneiric.g.Mcharacter.Name + texts["aa"] +
                    " " + enemy.GetType());
                string damage = Oneiric.g.Mcharacter.Attack(enemy);
                WriteOnHistory(Oneiric.g.Mcharacter.Name + texts["in"] +
                    " " + damage + texts["dm"]);
                if (enemy.ActualLife == 0)
                {
                    WriteOnHistory(enemy.GetType() + texts["de"]);
                    WriteOnHistory(Oneiric.g.Mcharacter.Name + texts["wn"]);
                    Item i = enemy.DropItem();
                    if (i != null)
                    {
                        Oneiric.g.Mcharacter.AddItem(i);
                        WriteOnHistory(texts["yg"] + " " + i.Name);
                    }
                    endBattle = true;
                }
                endPlayerTurn = true;
                break;
            case 1:
                break;
            case 2:
                endPlayerTurn = ShowItems();
                break;
            case 3:
                Oneiric.g.Mcharacter.Protect();
                protecting = true;
                break;
            case 4:
                if (Game.rand.Next(0,1)+1 == 1)
                {
                    endBattle = true;
                }
                endPlayerTurn = true;
                break;
        }
    }

    public List<string> LoadDrawItems()
    {
        List<string> dw = new List<string>();
        if (Oneiric.g.Mcharacter.GetInventory().Count > 0)
        {
            foreach (KeyValuePair<Item, byte> i in
                    Oneiric.g.Mcharacter.GetInventory())
            {
                if (i.Key is ConsumableItem)
                {
                    dw.Add(Oneiric.ItemsName[i.Key.Name.Substring(0, 2)]
                    + " x" + i.Value);
                }
            }

            return dw;
        }
        else
            return null;
    }

    public bool ShowItems()
    {
        int selected = 0;

        do
        {
            SdlHardware.Pause(100);
            List<string> drawItems = LoadDrawItems();
            if (drawItems == null)
            {
                SdlHardware.WriteHiddenText("NO HAY OBJETOS",
                    552, 422,
                    0x00, 0x00, 0x00,
                    Font28);
                SdlHardware.WriteHiddenText("NO HAY OBJETOS",
                    550, 420,
                    0xFF, 0xFF, 0xFF,
                    Font28);
            }
            else
            {
                short posX = 840;
                short posY = 420;

                int index = selected - 6;
                index = index < 0 ? 0 : index;

                SdlHardware.DrawHiddenImage(items, 780, 400);

                for (int i = 0; i < 10 && i < drawItems.Count - 1; i++)
                {
                    SdlHardware.WriteHiddenText(drawItems[index],
                        (short)(posX + 2), (short)(posY + 2),
                        0x00, 0x00, 0x00,
                        font16);
                    SdlHardware.WriteHiddenText(drawItems[index],
                        posX, posY,
                        0xFF, 0xFF, 0xFF,
                        font16);
                    posY += 30;
                    index++;
                }

                int minSelected = selected - 10;
                minSelected = minSelected < 10 ? 0 : minSelected;
                int maxSelected = minSelected + 10;
                maxSelected = minSelected < 10 ? index - 1 : maxSelected;

                SdlHardware.ShowHiddenScreen();
                if (SdlHardware.KeyPressed(SdlHardware.KEY_W) && selected >
                     minSelected)
                {
                    selected--;
                }
                else if (SdlHardware.KeyPressed(SdlHardware.KEY_S) && selected <
                    maxSelected)
                {
                    selected++;
                }
                else if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
                {
                    Oneiric.g.Mcharacter.UseItem(drawItems[selected].Substring(0, 2));
                    return true;
                }

                SdlHardware.DrawHiddenImage(selector2, 800, 418 + 30 * selected);
                SdlHardware.ShowHiddenScreen();
                SdlHardware.Pause(100);

            }
        } while (!SdlHardware.KeyPressed(SdlHardware.KEY_ESC));
        return false;
    }

    public void ShowHistory() {
        short posX = 30;
        short posY = 450;
        foreach (string s in history) {
            SdlHardware.WriteHiddenText(s,
                (short)(posX+2), (short)(posY+2),
                0x00, 0x00, 0x00,
                font16);
            SdlHardware.WriteHiddenText(s,
                posX, posY,
                0xFF, 0xFF, 0xFF,
                font16);
            posY += 20;
        }
    }
```

```
public void WriteOnHistory(string s)
{
   history.Enqueue(s);
   history.Dequeue();
   UpdateScreen();
   SdlHardware.Pause(1000);
}

public static void PrepareBattle()
{
   RandomEnemy(Game.rand.Next(0,15)+1);
   enemy.MoveTo(100,100);
}

public static NormalEnemy RandomEnemy(int random)
{
   switch (random)
   {
      case 1:
         enemy = new Brigthrooster();
         break;
      case 2:
         enemy = new Chimera();
         break;
      case 3:
         enemy = new Garuda();
         break;
      case 4:
         enemy = new Ghost();
         break;
      case 5:
         enemy = new Mimic();
         break;
      case 6:
         enemy = new Nightmare();
         break;
      case 7:
         enemy = new NightmareSoldier();
         break;
      case 8:
         enemy = new Ogre();
         break;
      case 9:
         enemy = new Orc();
         break;
      case 10:
         enemy = new Puppet();
         break;
      case 11:
         enemy = new Skeleton();
```

```
            break;
        case 12:
            enemy = new Succubus();
            break;
        case 13:
            enemy = new Vampire();
            break;
        case 14:
            enemy = new Werewolf();
            break;
        case 15:
            enemy = new Zombie();
            break;
    }

    return enemy;
}

public void DrawMenu()
{
    SdlHardware.DrawHiddenImage(Wallpaper, 0, 0);
    SdlHardware.DrawHiddenImage(menu, 650, 500);
    SdlHardware.DrawHiddenImage(historyI, 20, 425);
    SdlHardware.WriteHiddenText(texts["at"],
        682, 522,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["at"],
        680, 520,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["sk"],
       682, 562,
       0x00, 0x00, 0x00,
       Font28);
    SdlHardware.WriteHiddenText(texts["sk"],
        680, 560,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["it"],
       682, 602,
       0x00, 0x00, 0x00,
       Font28);
    SdlHardware.WriteHiddenText(texts["it"],
        680, 600,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["pt"],
       682, 642,
       0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(texts["pt"],
             680, 640,
             0xFF, 0xFF, 0xFF,
             Font28);
        SdlHardware.WriteHiddenText(texts["rn"],
            682, 682,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(texts["rn"],
             680, 680,
             0xFF, 0xFF, 0xFF,
             Font28);
        SdlHardware.DrawHiddenImage(selector, 670, 522 + 40 * option);
        SdlHardware.WriteHiddenText(Convert.ToString(
             Oneiric.g.Mcharacter.ActualLife),
            900, 250,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(Convert.ToString(
             Oneiric.g.Mcharacter.ActualLife),
             900, 250,
             0xFF, 0xFF, 0xFF,
             Font28);
        SdlHardware.WriteHiddenText(Convert.ToString(
             enemy.ActualLife),
            200, 375,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(Convert.ToString(
             enemy.ActualLife),
             200, 375,
             0xFF, 0xFF, 0xFF,
             Font28);
        SdlHardware.DrawHiddenImage(player, 900,200);
        enemy.DrawOnHiddenScreen();
    }
}
class Bosses : Enemy
{

}
class Brigthrooster : NormalEnemy
{
    public Brigthrooster()
    {
        LoadImage("data/images/enemies/normal/brightrooster.png");

        LifeIncreaser = 21;
        PmIncreaser = 17;
        DamageIncreaser = 5;
        DefenseIncreaser = 6;
        SpeedIncreaser = 16;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
using System;
using System.Collections.Generic;

abstract class Character : Sprite
{
    public int Level { get; set; }
    public int MaxiumLife { get; set; }
    public int ActualLife { get; set; }
    public int LifeIncreaser { get; set; }
    public int MaxiumPm { get; set; }
    public int ActualPm { get; set; }
    public int PmIncreaser { get; set; }
    public int Damage { get; set; }
    public int DamageIncreaser { get; set; }
    public int Defense { get; set; }
    public int DefenseIncreaser { get; set; }
    public int Speed { get; set; }
    public int SpeedIncreaser { get; set; }
    public int Lucky { get; set; }
    public List<Skill> Skills { get; set; }

    public Character() {}

    public string Attack(Character focus) {
        int damage = Damage +
            ((Damage/2) * Game.rand.Next(0, Lucky) + 1)
            - focus.Defense;
        damage = damage < 0 ? 1 : damage;
        focus.ActualLife -= damage;
        if (focus.ActualLife < 0)
            focus.ActualLife = 0;
        return damage.ToString();
    }
}
using System;
using System.Collections.Generic;
using System.IO;
```

```
class Chest : InteractibleElement
{
    protected int rarity;
    protected int maxItems;
    protected List<Item> items;
    public Chest(string image, int rarity)
        :base(image)
    {
        this.rarity = rarity;
        maxItems = Game.rand.Next(rarity, rarity*2 + 1);
        SdlHardware.Pause(40);
        items = new List<Item>();
        AddItems();
    }

    public void AddItems()
    {
        while (items.Count == 0)
        {
            try
            {
                StreamReader file = File.OpenText("data/langs/"+
                    Oneiric.Languages[Oneiric.Language].Substring(
                        0, 2).ToLower() + "/items.dat");
                string line = file.ReadLine();

                do
                {
                    line = file.ReadLine();

                    if (line != null)
                    {
                        string[] data = line.Split(';');

                        if (Convert.ToInt32(data[data.Length - 1]) == rarity)
                        {
                            if (Game.rand.Next(0, 4) == 1)
                            {
                                if (data[0] == "c")
                                {
                                                        items.Add(new  ConsumableItem(data[1],
Convert.ToInt32(data[2]),
                    Convert.ToInt32(data[3]), Convert.ToInt32(data[4]),
                    Convert.ToInt32(data[5]), Convert.ToInt32(data[6]),
                    Convert.ToInt32(data[7]), Convert.ToInt32(data[8]),
                    Convert.ToInt32(data[9])));
                                }
                                else if (data[0] == "e")
                                {
```

```
                        items.Add(new EquipableItem(data[1], Convert.ToInt32(data[2]),
                            Convert.ToInt32(data[3]), Convert.ToInt32(data[4]),
                            Convert.ToInt32(data[5]), Convert.ToInt32(data[6]),
                            Convert.ToInt32(data[7]), Convert.ToInt32(data[8])));
                    }
                    else
                    {
                        items.Add(new Item(data[1], Convert.ToInt32(data[2]),
                            Convert.ToInt32(data[3]), Convert.ToInt32(data[4]),
                            Convert.ToInt32(data[5]), Convert.ToInt32(data[6]),
                            Convert.ToInt32(data[7]), Convert.ToInt32(data[8])));
                    }

                }
            }
        }
    } while (line != null && items.Count < maxItems);

    file.Close();
}
catch (PathTooLongException)
{
    Oneiric.SaveLog("Path too long Error");
}
catch (FileNotFoundException)
{
    Oneiric.SaveLog("File Not Found");
}
catch (IOException e)
{
    Oneiric.SaveLog("IO Error: " + e);
}
catch (Exception e)
{
    Oneiric.SaveLog("Error: " + e);
}
}
}

public void Interactue(MainCharacter c)
{
    foreach (Item item in items)
    {
        c.AddItem(item);
    }
    items.Clear();
    LoadImage("data/images/map/computerOn.png");
}
}
class Chimera : NormalEnemy
```

```
{
   public Chimera()
   {
      LoadImage("data/images/enemies/normal/chimera.png");

      LifeIncreaser = 44;
      PmIncreaser = 15;
      DamageIncreaser = 8;
      DefenseIncreaser = 14;
      SpeedIncreaser = 5;

      MaxiumLife = LifeIncreaser * Level;
      MaxiumPm = PmIncreaser * Level;
      Damage = DamageIncreaser * Level;
      Defense = DefenseIncreaser * Level;
      Speed = SpeedIncreaser * Level;

      ActualLife = MaxiumLife;
   }
}
using System;

[Serializable]
class ConsumableItem : Item
{
   public int Heal { get; set; }
   public ConsumableItem(string name, int heal, int lfI, int pmI, int daI, int deI,
      int spI, int luI, int rarity)
      : base(name, lfI,pmI, daI, deI, spI, luI, rarity)
   {
      Heal = heal;
   }

   public override bool Use()
   {
      bool used = base.Use();
      if (Oneiric.g.Mcharacter.ActualLife != Oneiric.g.Mcharacter.MaxiumLife)
      {
         Oneiric.g.Mcharacter.Heal(Heal);
         used = true;
      }

      return used;
   }
}
class CreditScreen
{

}
class EffectSkill : Skill
```

```
{

}
using System;
using System.Collections.Generic;
using System.IO;

abstract class Enemy : Character
{
   protected static List<Item> droppableItems;

   public Enemy(){
      droppableItems = new List<Item>();
      LoadItems();
   }

   public Item DropItem()
   {
      if (Game.rand.Next(0, 2) == 1) {
         return droppableItems[Game.rand.Next(0, droppableItems.Count)];
      }

      return null;
   }

   public void LoadItems()
   {
      try
      {
         StreamReader file = File.OpenText("data/langs/" +
            Oneiric.Languages[Oneiric.Language].Substring(
               0, 2).ToLower() + "/items.dat");

         string line = "";

         do
         {
            line = file.ReadLine();

            if (line != null)
            {
               string[] data = line.Split(';');
               if (data[0] == "c")
               {
                                    droppableItems.Add(new  ConsumableItem(data[1],
Convert.ToInt32(data[2]),
                    Convert.ToInt32(data[3]), Convert.ToInt32(data[4]),
                    Convert.ToInt32(data[5]), Convert.ToInt32(data[6]),
                    Convert.ToInt32(data[7]), Convert.ToInt32(data[8]),
                    Convert.ToInt32(data[9])));
```

```
                }
                else if (data[0] == "e")
                {
                                        droppableItems.Add(new  EquipableItem(data[1],
Convert.ToInt32(data[2]),
                    Convert.ToInt32(data[3]), Convert.ToInt32(data[4]),
                    Convert.ToInt32(data[5]), Convert.ToInt32(data[6]),
                    Convert.ToInt32(data[7]), Convert.ToInt32(data[8])));
                }
                else
                {
                  droppableItems.Add(new Item(data[1], Convert.ToInt32(data[2]),
                    Convert.ToInt32(data[3]), Convert.ToInt32(data[4]),
                    Convert.ToInt32(data[5]), Convert.ToInt32(data[6]),
                    Convert.ToInt32(data[7]), Convert.ToInt32(data[8])));
                }

              }
            } while (line != null);
          }
          catch (PathTooLongException)
          {
            Oneiric.SaveLog("Path too long Error");
          }
          catch (FileNotFoundException)
          {
            Oneiric.SaveLog("File Not Found");
          }
          catch (IOException e)
          {
            Oneiric.SaveLog("IO Error: " + e);
          }
          catch (Exception e)
          {
            Oneiric.SaveLog("Error: " + e);
          }
        }
}
using System;

[Serializable]
class EquipableItem : Item
{
    public EquipableItem(string name, int lfI, int pmI, int daI, int deI,
        int spI, int luI, int rarity)
        : base(name, lfI, pmI, daI, deI, spI, luI, rarity)
    {

    }
}
```

```
using System;
using Tao.Sdl;

[Serializable]
class Font
{
   private IntPtr internalPointer;

   public Font(string fileName, short sizePoints)
   {
      Load(fileName, sizePoints);
   }

   public void Load(string fileName, short sizePoints)
   {
      internalPointer = SdlTtf.TTF_OpenFont(fileName, sizePoints);
      if (internalPointer == IntPtr.Zero)
         SdlHardware.FatalError("Font not found: " + fileName);
   }

   public IntPtr GetPointer()
   {
      return internalPointer;
   }
}
using System;
using System.Collections.Generic;

class Game
{
   public MainCharacter Mcharacter { get; set; }
   public Room Groom { get; set; }
   protected bool finished;
   protected Font font18;
   public static Random rand;
   public int Steps { get; set; }
   protected int randMax = 350;
   protected int randMin = 50;
   protected byte countTimer;
   public long Time { get; set; }
   protected GameMenuScreen gm;
   protected BattleScreen bs;
   protected GameOverScreen go;
   public static int AverageEnemyLevel { get; set; }
   public bool CanFight { get; set; }

   public Game()
   {
      Mcharacter = new MainCharacter();
      finished = false;
```

```
    font18 = new Font("data/fonts/Joystix.ttf", 18);
    Groom = new Room();
    rand = new Random();
    Steps = rand.Next(randMin,randMax);
    Time = 0;
    countTimer = 0;
    AverageEnemyLevel = 6;
    CanFight = true;
    gm = new GameMenuScreen();
}

void UpdateScreen()
{
    SdlHardware.ClearScreen();
    Groom.DrawOnHiddenScreen();

    Mcharacter.DrawOnHiddenScreen();

    SdlHardware.ShowHiddenScreen();
}

void CheckInput()
{
    if (SdlHardware.KeyPressed(SdlHardware.KEY_D) && Mcharacter.GetX()
        >= Groom.MaxRight)
    {
        Groom.ActualCol++;
        Groom.UpdateScreenMap(Groom.ActualCol, Groom.ActualRow);
        Mcharacter.MoveTo(0, Mcharacter.GetY());
        Steps--;
    }
    else if (SdlHardware.KeyPressed(SdlHardware.KEY_D))
    {
        if (Groom.CanMoveTo(Mcharacter.GetX() + Mcharacter.GetSpeedX(),
                Mcharacter.GetY(),
                Mcharacter.GetX() + Mcharacter.GetWidth() +
                Mcharacter.GetSpeedX(),
                Mcharacter.GetY() + Mcharacter.GetHeight()))
        {
            Mcharacter.MoveRight();
            Steps--;
        }
        else
        {
            Mcharacter.NextFrame();
        }
        Mcharacter.ChangeDirection(Sprite.RIGHT);
    }
    else if (SdlHardware.KeyPressed(SdlHardware.KEY_A) &&
        Mcharacter.GetX() <= 0)
```

```
{
   Groom.ActualCol--;
   Groom.UpdateScreenMap(Groom.ActualCol, Groom.ActualRow);
   Mcharacter.MoveTo(Groom.MaxRight, Mcharacter.GetY());
   Steps--;
}
else if (SdlHardware.KeyPressed(SdlHardware.KEY_A))
{
   if (Groom.CanMoveTo(Mcharacter.GetX() - Mcharacter.GetSpeedX(),
         Mcharacter.GetY(),
         Mcharacter.GetX() + Mcharacter.GetWidth() -
         Mcharacter.GetSpeedX(),
         Mcharacter.GetY() + Mcharacter.GetHeight()))
   {
      Mcharacter.MoveLeft();
      Steps--;
   }
   else
   {
      Mcharacter.NextFrame();
   }
   Mcharacter.ChangeDirection(Sprite.LEFT);
}
else if (SdlHardware.KeyPressed(SdlHardware.KEY_W) &&
   Mcharacter.GetY() <= Groom.GetTopMargin())
{
   Groom.ActualRow--;
   Groom.UpdateScreenMap(Groom.ActualCol, Groom.ActualRow);
   Mcharacter.MoveTo(Mcharacter.GetX(), Groom.MaxDown);
   Steps--;
}
else if (SdlHardware.KeyPressed(SdlHardware.KEY_W))
{
   if (Groom.CanMoveTo(Mcharacter.GetX(),
         Mcharacter.GetY() - Mcharacter.GetSpeedY(),
         Mcharacter.GetX() + Mcharacter.GetWidth(),
         Mcharacter.GetY() + Mcharacter.GetHeight() -
         Mcharacter.GetSpeedY()))
   {
      Mcharacter.MoveUp();
      Steps--;
   }
   else
   {
      Mcharacter.NextFrame();
   }
   Mcharacter.ChangeDirection(Sprite.UP);
}
else if (SdlHardware.KeyPressed(SdlHardware.KEY_S) &&
   Mcharacter.GetY() >= Groom.MaxDown)
```

```
    {
        Groom.ActualRow++;
        Groom.UpdateScreenMap(Groom.ActualCol, Groom.ActualRow);
        Mcharacter.MoveTo(Mcharacter.GetX(), Groom.GetTopMargin());
        Steps--;
    }
    else if (SdlHardware.KeyPressed(SdlHardware.KEY_S))
    {
        if (Groom.CanMoveTo(Mcharacter.GetX(),
                Mcharacter.GetY() + Mcharacter.GetSpeedY(),
                Mcharacter.GetX() + Mcharacter.GetWidth(),
                Mcharacter.GetY() + Mcharacter.GetHeight() +
                Mcharacter.GetSpeedY()))
        {
            Mcharacter.MoveDown();
            Steps--;
        }
        else
        {
            Mcharacter.NextFrame();
        }
        Mcharacter.ChangeDirection(Sprite.DOWN);
    }
    else if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
    {
        SdlHardware.Pause(100);
        foreach (Chest c in Groom.chests)
        {
            if (c.canInteractue(Mcharacter))
            {
                c.Interactue(Mcharacter);
            }
        }
    }

    if (SdlHardware.KeyPressed(SdlHardware.KEY_I))
    {
        if (gm.Run(Groom, Mcharacter) == 5)
        {
            finished = true;
        }
    }
}

void CheckFights()
{
    if (Steps == 0)
    {
        bs = new BattleScreen();
        bs.Run();
        if (Mcharacter.ActualLife == 0)
        {
            if (CanFight)
            {
                go = new GameOverScreen();
                go.Run();
                finished = true;
            }
            else
            {
                Steps = RandomSteps();
            }

        }
        Steps = RandomSteps();
    }
}

int RandomSteps()
{
    return rand.Next(randMin, randMax);
}

void Timer()
{
    if (countTimer == 25)
    {
        Time++;
        countTimer = 0;
    }

    countTimer++;
}

void PauseUntilNextFrame()
{
    SdlHardware.Pause(40); //(40 ms = 25 fps)
}

public void Run()
{
    do
    {
        UpdateScreen();

        CheckInput();

        CheckFights();

        Timer();
```

```
            PauseUntilNextFrame();
        }
        while (!finished);
    }
}
using System.Collections.Generic;
using System;

[Serializable]
class GameMenuScreen : Screen
{
    protected Image selector, secondSelector, saveBackground, greyBackground, face;
    protected Image[] icons;
    protected int option;
    protected Font font72;

    const int YCURSOR_MAX = 6;
    const int YCURSOR_MIN = 0;
    const int TOTAL_ICONS = 7;

    public GameMenuScreen()
        : base(new Image("data/images/other/menuInGame.png"),
            new Font("data/fonts/Joystix.ttf", 28))
    {
        option = 0;
        selector = new Image("data/images/other/selector2.png");
        secondSelector = new Image("data/images/other/selector2.png");
        saveBackground = new Image("data/images/other/loadGame.png");
        greyBackground = new Image("data/images/other/greyBackground.png");
        face = new Image("data/images/player/Face.png");
        icons = new Image[TOTAL_ICONS];
        icons[0] = new Image("data/images/other/icons/Life.png");
        icons[1] = new Image("data/images/other/icons/Laptoop.png");
        icons[2] = new Image("data/images/other/icons/Damage.png");
        icons[3] = new Image("data/images/other/icons/Defense.png");
        icons[4] = new Image("data/images/other/icons/XP.png");
        icons[5] = new Image("data/images/other/icons/Speed.png");
        icons[6] = new Image("data/images/other/icons/Lucky.png");
        font72 = new Font("data/fonts/Joystix.ttf", 72);
        texts = new Dictionary<string, string>();
    }

    public int Run(Room room, MainCharacter character)
    {
        option = 0;
        LoadText(Oneiric.Languages[Oneiric.Language], "gameMenu");

        do
        {
```

```
        SdlHardware.ClearScreen();
        room.DrawOnHiddenScreen();
        character.DrawOnHiddenScreen();
        DrawMenu();
        SdlHardware.ShowHiddenScreen();
        if (SdlHardware.KeyPressed(SdlHardware.KEY_W) && option >
           YCURSOR_MIN)
        {
           option--;
        }
        else if (SdlHardware.KeyPressed(SdlHardware.KEY_S) && option <
           YCURSOR_MAX)
        {
           option++;
        }
        else if (SdlHardware.KeyPressed(SdlHardware.KEY_ESC))
        {
           option = YCURSOR_MAX;
        }
        else if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
        {
           int optionSelected = ChoosedOption();
           if (optionSelected == 5 || optionSelected == YCURSOR_MAX)
           {
              return ChoosedOption();
           }
           SdlHardware.Pause(200);
        }
        SdlHardware.Pause(100);
     }
     while (!SdlHardware.KeyPressed(SdlHardware.KEY_ESC));
     return 0;
  }

  public int ChoosedOption()
  {
     int value = 0;
     switch (option)
     {
        case 0:
           ShowParty();
           value = 0;
           break;
        case 1:
           SdlHardware.Pause(200);
           ShowInventory();
           value = 1;
           break;
        case 2:
           value = 2;
```

```
            break;
        case 3:
            SaveMenu();
            value = 3;
            break;
        case 4:
            Oneiric.os.Run();
            Oneiric.SaveOptions();
            value = 4;
            break;
        case 5:
            value = 5;
            break;
        case 6:
            value = 6;
            break;
    }

    return value;
}

public void ShowParty()
{
    SdlHardware.Pause(100);
    do
    {
        SdlHardware.DrawHiddenImage(greyBackground, 0, 0);
        SdlHardware.DrawHiddenImage(face, 620, 200);
        SdlHardware.DrawHiddenImage(icons[0], 212, 200);
        SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.ActualLife.ToString()
            + " / " + Oneiric.g.Mcharacter.MaxiumLife.ToString(),
            252, 202,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.ActualLife.ToString()
            + " / " + Oneiric.g.Mcharacter.MaxiumLife.ToString(),
            250, 200,
            0xFF, 0xFF, 0xFF,
            Font28);
        SdlHardware.DrawHiddenImage(icons[1], 212, 250);
        SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.ActualPm.ToString()
            + " / " + Oneiric.g.Mcharacter.MaxiumPm.ToString(),
            252, 252,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.ActualPm.ToString()
            + " / " + Oneiric.g.Mcharacter.MaxiumPm.ToString(),
            250, 250,
            0xFF, 0xFF, 0xFF,
            Font28);
```

```
SdlHardware.DrawHiddenImage(icons[2], 212, 300);
SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.Damage.ToString(),
    252, 302,
    0x00, 0x00, 0x00,
    Font28);
SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.Damage.ToString(),
    250, 300,
    0xFF, 0xFF, 0xFF,
    Font28);
SdlHardware.DrawHiddenImage(icons[3], 212, 350);
SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.Defense.ToString(),
    252, 352,
    0x00, 0x00, 0x00,
    Font28);
SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.Defense.ToString(),
    250, 350,
    0xFF, 0xFF, 0xFF,
    Font28);
SdlHardware.DrawHiddenImage(icons[5], 212, 400);
SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.Speed.ToString(),
    252, 402,
    0x00, 0x00, 0x00,
    Font28);
SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.Speed.ToString(),
    250, 400,
    0xFF, 0xFF, 0xFF,
    Font28);
SdlHardware.DrawHiddenImage(icons[6], 212, 450);
SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.Lucky.ToString(),
    252, 452,
    0x00, 0x00, 0x00,
    Font28);
SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.Lucky.ToString(),
    250, 450,
    0xFF, 0xFF, 0xFF,
    Font28);
SdlHardware.WriteHiddenText(Oneiric.g.Mcharacter.Speed.ToString(),
    250, 400,
    0xFF, 0xFF, 0xFF,
    Font28);
SdlHardware.WriteHiddenText("ESC -->",
    602, 622,
    0x00, 0x00, 0x00,
    Font28);
SdlHardware.WriteHiddenText("ESC -->",
    600, 620,
    0xFF, 0xFF, 0xFF,
    Font28);
SdlHardware.ShowHiddenScreen();
```

```
        SdlHardware.Pause(100);
    } while (!SdlHardware.KeyPressed(SdlHardware.KEY_ESC));
}

public List<string> LoadDrawItems() {
    List<string> dw = new List<string>();
    if (Oneiric.g.Mcharacter.GetInventory().Count > 0)
    {
        foreach (KeyValuePair<Item, byte> i in
                Oneiric.g.Mcharacter.GetInventory())
        {
            dw.Add(Oneiric.ItemsName[i.Key.Name.Substring(0, 2)]
                + " x" + i.Value);
        }

        return dw;
    }
    else
        return null;
}

public void ShowInventory()
{
    int selected = 0;

    do
    {
        List<string> drawItems = LoadDrawItems();
        if (drawItems == null)
        {
            SdlHardware.WriteHiddenText("NO HAY OBJETOS",
                552, 422,
                0x00, 0x00, 0x00,
                Font28);
            SdlHardware.WriteHiddenText("NO HAY OBJETOS",
                550, 420,
                0xFF, 0xFF, 0xFF,
                Font28);
        }
        else
        {
            short posX = 200;
            short posY = 230;

            int index = selected - 10;
            index = index < 0 ? 0 : index;

            SdlHardware.DrawHiddenImage(greyBackground, 0, 0);
            SdlHardware.DrawHiddenImage(selector, 160, 232 + 30 * selected);
            for (int i = 0; i < 10 && i < drawItems.Count - 1; i++)
            {
                SdlHardware.WriteHiddenText(drawItems[index],
                        (short)(posX + 2), (short)(posY + 2),
                        0x00, 0x00, 0x00,
                        Font28);
                SdlHardware.WriteHiddenText(drawItems[index],
                    posX, posY,
                    0xFF, 0xFF, 0xFF,
                    Font28);
                posY += 30;
                index++;
            }

            int minSelected = selected - 10;
            minSelected = minSelected < 10 ? 0 : minSelected;
            int maxSelected = minSelected + 10;
            maxSelected = minSelected < 10 ? index - 1 : maxSelected;

            SdlHardware.ShowHiddenScreen();
            if (SdlHardware.KeyPressed(SdlHardware.KEY_W) && selected >
                 minSelected)
            {
                selected--;
            }
            else if (SdlHardware.KeyPressed(SdlHardware.KEY_S) && selected <
                maxSelected)
            {
                selected++;
            }
            else if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
            {
                Oneiric.g.Mcharacter.UseItem(drawItems[selected].Substring(0, 2));
            }
            SdlHardware.Pause(100);
            SdlHardware.ShowHiddenScreen();
        }
    } while (!SdlHardware.KeyPressed(SdlHardware.KEY_ESC));
}

public string SelectedOption()
{
    string rt = "";
    switch (option)
    {
        case 0:
            rt = "eq";
            break;
        case 1:
            rt = "in";
            break;
        case 2:
            rt = "sn";
            break;
        case 3:
            rt = "sv";
            break;
        case 4:
            rt = "op";
            break;
        case 5:
            rt = "tt";
            break;
        case 6:
            rt = "cl";
            break;
    }

    return rt;
}

public void DrawMenu()
{
    SdlHardware.DrawHiddenImage(Wallpaper, 0, 0);
    SdlHardware.WriteHiddenText(texts[SelectedOption()],
        352, 132,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts[SelectedOption()],
        350, 132,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["mn"],
        962, 132,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["mn"],
        960, 130,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["eq"],
        922, 182,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["eq"],
        920, 180,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["in"],
        922, 252,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["in"],
        920, 250,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["sn"],
        922, 322,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["sn"],
        920, 320,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["sv"],
        922, 392,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["sv"],
        920, 390,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["op"],
        922, 462,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["op"],
        920, 460,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["tt"],
        922, 532,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["tt"],
        920, 530,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["cl"],
        922, 602,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["cl"],
        920, 600,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.DrawHiddenImage(selector, 890, 183 + 70 * option);
}

public void SaveMenu()
{
        int optionSave = 0;
        const int YSECONDCURSOR_MAX = 2;
        const int YSECONDCURSOR_MIN = 0;
        SdlHardware.Pause(100);
        do
        {
            SdlHardware.DrawHiddenImage(greyBackground, 0, 0);
            SdlHardware.DrawHiddenImage(saveBackground, -100, 0);
            SdlHardware.DrawHiddenImage(selector, 200, 270 + 120 * optionSave);
            SdlHardware.ShowHiddenScreen();
            if (SdlHardware.KeyPressed(SdlHardware.KEY_W) && optionSave >
                 YSECONDCURSOR_MIN)
            {
                optionSave--;
            }
            else if (SdlHardware.KeyPressed(SdlHardware.KEY_S) && optionSave <
                YSECONDCURSOR_MAX)
            {
                optionSave++;
            }
            else if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
            {
                Oneiric.SaveGame("data/savedGames/" + (optionSave + 1) + "_game.save");
            }
            SdlHardware.Pause(100);
        } while (!SdlHardware.KeyPressed(SdlHardware.KEY_ESC));
    }
}
class GameOverScreen : Screen
{
    protected Font font72;

    public GameOverScreen()
        : base(new Image("data/images/other/welcome.jpg"),
            new Font("data/fonts/Joystix.ttf", 28))
    {
        font72 = new Font("data/fonts/Joystix.ttf", 72);
    }

    public void Run()
    {
        LoadText(Oneiric.Languages[Oneiric.Language], "gameOver");
        SdlHardware.Pause(100);
        do
        {
            SdlHardware.ClearScreen();
            DrawMenu();
            SdlHardware.ShowHiddenScreen();
            if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
            {
                return;
            }
            SdlHardware.Pause(100);
        }
        while (true);
        //The loop ends when an option is choosed.
    }

    public void DrawMenu()
    {

        SdlHardware.WriteHiddenText(texts["go"],
           442, 102,
           0x00, 0x00, 0x00,
           font72);
        SdlHardware.WriteHiddenText(texts["go"],
            440, 100,
            0xFF, 0xFF, 0xFF,
            font72);
        SdlHardware.WriteHiddenText(texts["pe"],
           242, 302,
           0x00, 0x00, 0x00,
           Font28);
        SdlHardware.WriteHiddenText(texts["pe"],
            240, 300,
            0xFF, 0xFF, 0xFF,
            Font28);
    }
}
class Garuda : NormalEnemy
{
    public Garuda()
    {
        LoadImage("data/images/enemies/normal/garuda.png");

        LifeIncreaser = 32;
        PmIncreaser = 21;
        DamageIncreaser = 12;
        DefenseIncreaser = 4;
        SpeedIncreaser = 11;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
```

```
class Ghost : NormalEnemy
{
    public Ghost()
    {
        LoadImage("data/images/enemies/normal/ghost.png");

        LifeIncreaser = 28;
        PmIncreaser = 18;
        DamageIncreaser = 14;
        DefenseIncreaser = 8;
        SpeedIncreaser = 6;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
using System.Collections.Generic;

class HelpScreen : Screen
{
    protected Image controls;
    protected Font font72, font28;

    public HelpScreen()
        : base(new Image("data/images/other/welcome.jpg"),
            new Font("data/fonts/Joystix.ttf", 28))
    {
        controls = new Image("data/images/other/controls.png");
        font72 = new Font("data/fonts/Joystix.ttf", 72);
        font28 = new Font("data/fonts/Joystix.ttf", 28);
        texts = new Dictionary<string, string>();
    }

    public void Run()
    {
        LoadText(Oneiric.Languages[Oneiric.Language], "helpMenu");
        SdlHardware.Pause(100);
        do
        {
            SdlHardware.ClearScreen();
            DrawMenu();
            SdlHardware.ShowHiddenScreen();
            if  (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
            {
                return;
            }
            SdlHardware.Pause(100);
        }
        while (true);
        //The loop ends when an option is choosed.
    }

    public void DrawMenu()
    {
        SdlHardware.DrawHiddenImage(Wallpaper, 0, 0);
        SdlHardware.DrawHiddenImage(controls, 30, 100);

        SdlHardware.WriteHiddenText(texts["hp"],
           442, 102,
           0x00, 0x00, 0x00,
           font72);
        SdlHardware.WriteHiddenText(texts["hp"],
            440, 100,
            0xFF, 0xFF, 0xFF,
            font72);

        SdlHardware.WriteHiddenText(texts["mv"],
           132, 552,
           0x00, 0x00, 0x00,
           font28);
        SdlHardware.WriteHiddenText(texts["mv"],
            130, 550,
            0xFF, 0xFF, 0xFF,
            font28);
        SdlHardware.WriteHiddenText(texts["iv"],
           512, 552,
           0x00, 0x00, 0x00,
           font28);
        SdlHardware.WriteHiddenText(texts["iv"],
            510, 550,
            0xFF, 0xFF, 0xFF,
            font28);
        SdlHardware.WriteHiddenText(texts["it"],
           872, 552,
           0x00, 0x00, 0x00,
           font28);
        SdlHardware.WriteHiddenText(texts["it"],
            870, 550,
            0xFF, 0xFF, 0xFF,
            font28);
        SdlHardware.WriteHiddenText(texts["mg"],
           322, 652,
           0x00, 0x00, 0x00,
           font28);
        SdlHardware.WriteHiddenText(texts["mg"],
            320, 650,
            0xFF, 0xFF, 0xFF,
            font28);
    }
}
using System;
using Tao.Sdl;
[Serializable]
class Image
{
    private IntPtr internalPointer;

    public Image(string fileName)  // Constructor
    {
        Load(fileName);
    }

    public void Load(string fileName)
    {
        internalPointer = SdlImage.IMG_Load(fileName);
        if (internalPointer == IntPtr.Zero)
        {
            Oneiric.SaveLog("Image not found: " + fileName);
            SdlHardware.FatalError("Image not found: " + fileName);
        }
    }


    public IntPtr GetPointer()
    {
        return internalPointer;
    }
}
using System;

abstract class InteractibleElement : Sprite
{
    public InteractibleElement(string image)
        :base(image)
    {
    }

    public bool canInteractue(MainCharacter c) {
        //Console.WriteLine(c.GetX() + " - " + x);
        if (((c.GetX() >= x && c.GetX() <= x + 48) ||
            (c.GetX() + 37 >= x && c.GetX() + 37 <= x + 48)) &&
            c.GetY() == y + 48)
            return true;
        else
            return false;
    }
}
using System;

[Serializable]
class Item : Sprite
{
    public string Name { get; set; }
    public int LifeIncreaser { get; set; }
    public int PmIncreaser { get; set; }
    public int DamageIncreaser { get; set; }
    public int DefenseIncreaser { get; set; }
    public int SpeedIncreaser { get; set; }
    public int LuckyIncreaser { get; set; }
    public int Rarity { get; set; }

    public Item(string name, int lfI, int pmI, int daI,
        int deI, int spI, int luI, int rarity)
    {
        Name = name;
        LifeIncreaser = lfI;
        PmIncreaser = pmI;
        DamageIncreaser = daI;
        DefenseIncreaser = deI;
        SpeedIncreaser = spI;
        LuckyIncreaser = luI;
        Rarity = rarity;
    }

    public override bool Equals(Object obj)
    {
        Item i = (Item)obj;
        return i.Name.Equals(Name);
    }

    public override int GetHashCode()
    {
        return Name.GetHashCode();
    }

    public virtual bool Use() {
        bool used = false;
        if (LifeIncreaser != 0)
        {
            Oneiric.g.Mcharacter.MaxiumLife += LifeIncreaser;
            used = true;
        }

        if (PmIncreaser != 0)
        {
            Oneiric.g.Mcharacter.MaxiumPm += PmIncreaser;
            used = true;
        }

        if (DamageIncreaser != 0)
        {
            Oneiric.g.Mcharacter.Damage += DamageIncreaser;
            used = true;
        }

        if (DefenseIncreaser != 0)
        {
            Oneiric.g.Mcharacter.Defense += DefenseIncreaser;
            used = true;
        }

        if (SpeedIncreaser != 0)
        {
            Oneiric.g.Mcharacter.Speed += SpeedIncreaser;
            used = true;
        }

        if (LuckyIncreaser != 0)
        {
            Oneiric.g.Mcharacter.Lucky += LuckyIncreaser;
            used = true;
        }

        return used;
    }
}
using System;
using System.Collections.Generic;
using System.IO;

class LoadGamesScreen : Screen
{
    protected Image selector, realWallpaper;
    protected int option;
    protected Font font72, font12;

    const int YCURSOR_MAX = 3;
    const int YCURSOR_MIN = 0;

    public LoadGamesScreen()
        : base(new Image("data/images/other/loadGame.png"),
            new Font("data/fonts/Joystix.ttf", 28))
    {
        option = 0;
        selector = new Image("data/images/other/selector.png");
        realWallpaper = new Image("data/images/other/welcome.jpg");
        font72 = new Font("data/fonts/Joystix.ttf", 72);
        font12 = new Font("data/fonts/Joystix.ttf", 12);
        texts = new Dictionary<string, string>();
    }

    public int GetChosenOption()
    {
        return option;
    }

    public int Run()
    {
        option = 0;
        LoadText(Oneiric.Languages[Oneiric.Language], "loadSaveMenu");
        SdlHardware.Pause(100);
        do
        {
            SdlHardware.ClearScreen();
            DrawMenu();
            SdlHardware.ShowHiddenScreen();
            if (SdlHardware.KeyPressed(SdlHardware.KEY_W) && option >
                YCURSOR_MIN)
            {
                option--;
            }
            else if (SdlHardware.KeyPressed(SdlHardware.KEY_S) && option <
                YCURSOR_MAX)
            {
                option++;
            }
            else if (SdlHardware.KeyPressed(SdlHardware.KEY_ESC))
            {
                option = YCURSOR_MAX;
            }
            else if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
            {
                string nameFile = "data/savedGames/" + (option + 1) +
                        "_game.save";
                if (option == YCURSOR_MAX)
                {
                    return 1;
                }
                else
                {
                    if (!File.Exists(nameFile))
                    {
                        Oneiric.SaveLog("Can't load the game. File Not Found "+
                            nameFile);
                    }
                else
                {
                    Oneiric.LoadGame(nameFile);
                    return 0;
                }
            }

        }
        SdlHardware.Pause(100);
    }
    while (true);
    //The loop ends when an option is choosed.
    }

    public void DrawMenu()
    {
        SdlHardware.DrawHiddenImage(realWallpaper, 0, 0);
        SdlHardware.DrawHiddenImage(Wallpaper, 0, 0);
        SdlHardware.WriteHiddenText(texts["lg"],
           202, 102,
           0x00, 0x00, 0x00,
           font72);
        SdlHardware.WriteHiddenText(texts["lg"],
            200, 100,
            0xFF, 0xFF, 0xFF,
            font72);
        SdlHardware.WriteHiddenText("1",
           372, 267,
           0x00, 0x00, 0x00,
           Font28);
        SdlHardware.WriteHiddenText("1",
            370, 265,
            0xFF, 0xFF, 0xFF,
            Font28);

        if (File.Exists("data/savedGames/1_game.save"))
        {
            string time = CalculateTime(
                Oneiric.GetTime("data/savedGames/1_game.save"));

            SdlHardware.WriteHiddenText(texts["gt"] + time,
                592, 302,
                0x00, 0x00, 0x00,
                font12);
            SdlHardware.WriteHiddenText(texts["gt"] + time,
                590, 300,
                0xFF, 0xFF, 0xFF,
                font12);
        }
        else
```

```
{
    SdlHardware.WriteHiddenText(texts["nd"],
        482, 267,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["nd"],
        480, 265,
        0xFF, 0xFF, 0xFF,
        Font28);
}
SdlHardware.WriteHiddenText("2",
   372, 387,
   0x00, 0x00, 0x00,
   Font28);
SdlHardware.WriteHiddenText("2",
    370, 385,
    0xFF, 0xFF, 0xFF,
    Font28);
if (File.Exists("data/savedGames/2_game.save"))
{
    string time = CalculateTime(
        Oneiric.GetTime("data/savedGames/2_game.save"));

    SdlHardware.WriteHiddenText(texts["gt"] + time,
        592, 422,
        0x00, 0x00, 0x00,
        font12);
    SdlHardware.WriteHiddenText(texts["gt"] + time,
        590, 420,
        0xFF, 0xFF, 0xFF,
        font12);
}
else
{
    SdlHardware.WriteHiddenText(texts["nd"],
        482, 387,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["nd"],
        480, 385,
        0xFF, 0xFF, 0xFF,
        Font28);
}
SdlHardware.WriteHiddenText("3",
   372, 497,
   0x00, 0x00, 0x00,
   Font28);
SdlHardware.WriteHiddenText("3",
    370, 495,
    0xFF, 0xFF, 0xFF,
            Font28);
        if (File.Exists("data/savedGames/3_game.save"))
        {
            string time = CalculateTime(
                Oneiric.GetTime("data/savedGames/3_game.save"));

            SdlHardware.WriteHiddenText(texts["gt"] + time,
                592, 532,
                0x00, 0x00, 0x00,
                font12);
            SdlHardware.WriteHiddenText(texts["gt"] + time,
                590, 530,
                0xFF, 0xFF, 0xFF,
                font12);
        }
        else
        {
            SdlHardware.WriteHiddenText(texts["nd"],
                482, 497,
                0x00, 0x00, 0x00,
                Font28);
            SdlHardware.WriteHiddenText(texts["nd"],
                480, 495,
                0xFF, 0xFF, 0xFF,
                Font28);
        }
        SdlHardware.WriteHiddenText(texts["bc"],
            502, 618,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(texts["bc"],
            500, 616,
            0xFF, 0xFF, 0xFF,
            Font28);
        SdlHardware.DrawHiddenImage(selector, option != YCURSOR_MAX? 235:430,
            250 + 115 * option);
    }

    public string CalculateTime(long seconds)
    {
        long hours = (seconds / 3600);
        long minutes = ((seconds - hours * 3600) / 60);
        long secs = seconds - (hours * 3600 + minutes * 60);
        return hours.ToString() + ":" + minutes.ToString() + ":" + secs.ToString();
    }
}
using System;
using System.Collections.Generic;

class MainCharacter : Character
```

```
{
    public Dictionary<Item,byte> Inventory { get; set; }
    public List<Weapon> Weapons { get; set; }
    public string Name { get; }

    public MainCharacter()
    {
        LoadSequence(RIGHT,
            new string[] { "data/images/Player/Right_1.png",
                "data/images/Player/Right_2.png",
                "data/images/Player/Right_3.png"  });
        LoadSequence(LEFT,
            new string[] { "data/images/Player/Left_1.png",
                "data/images/Player/Left_2.png",
                "data/images/Player/Left_3.png"});
        LoadSequence(UP,
            new string[] { "data/images/Player/Up_1.png",
                "data/images/Player/Up_2.png",
                "data/images/Player/Up_3.png" });
        LoadSequence(DOWN,
            new string[] { "data/images/Player/Down_1.png",
                "data/images/Player/Down_2.png",
                "data/images/Player/Down_3.png" });
        currentDirection = UP;
        x = 240;
        y = 320;
        xSpeed = ySpeed = 8;
        width = 37;
        height = 47;
        Inventory = new Dictionary<Item, byte>();
        Name = "Coco";
        Level = 1;
        LifeIncreaser = 25;
        PmIncreaser = 20;
        DamageIncreaser = 9;
        DefenseIncreaser = 9;
        SpeedIncreaser = 5;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
        ActualPm = MaxiumPm;
    }

    public Dictionary<Item,byte> GetInventory() { return Inventory; }
    public void MoveRight()
    {
        x += xSpeed;
        NextFrame();
    }

    public void MoveLeft()
    {
        x -= xSpeed;
        NextFrame();
    }

    public void MoveUp()
    {
        y -= ySpeed;
        NextFrame();
    }

    public void MoveDown()
    {
        y += ySpeed;
        NextFrame();
    }

    public bool AddItem(Item i)
    {
        bool added = false;
        if (Inventory.ContainsKey(i))
        {
            if (Inventory[i] < 255)
            {
                Inventory[i] += 1;
                added = true;
            }
        }
        else
        {
            Inventory.Add(i,1);
            added = true;
        }

        return added;
    }

    public void UseItem(string itemName) {
        Item i = null;
        foreach(KeyValuePair<Item, byte> it in Inventory) {
            if (it.Key.Name.Substring(0,2) == itemName)
            {
                i = it.Key;
```

```
                break;
            }
        }

        if (i is ConsumableItem)
        {
            bool used = i.Use();
            if (used)
                Inventory[i] -= 1;

            if (Inventory[i] == 0)
                Inventory.Remove(i);
        }
    }

    public int Heal(int heal) {
        int healedCuantity;
        if (ActualLife + heal > MaxiumLife) {
            healedCuantity = MaxiumLife - ActualLife;
            ActualLife = MaxiumLife;
        }
        else {
            healedCuantity = heal;
            ActualLife += heal;
        }

        return healedCuantity;
    }

    public void Protect()
    {
        Defense *= 2;
    }
}
class Mimic : NormalEnemy
{
    public Mimic()
    {
        LoadImage("data/images/enemies/normal/mimic.png");

        LifeIncreaser = 60;
        PmIncreaser = 1;
        DamageIncreaser = 9;
        DefenseIncreaser = 35;
        SpeedIncreaser = 1;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
class Nightmare : NormalEnemy
{
    public Nightmare()
    {
        LoadImage("data/images/enemies/normal/nightmare.png");

        LifeIncreaser = 17;
        PmIncreaser = 9;
        DamageIncreaser = 19;
        DefenseIncreaser = 9;
        SpeedIncreaser = 3;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
class NightmareSoldier : NormalEnemy
{
    public NightmareSoldier()
    {
        LoadImage("data/images/enemies/normal/nightmaresoldier.png");

        LifeIncreaser = 35;
        PmIncreaser = 4;
        DamageIncreaser = 10;
        DefenseIncreaser = 10;
        SpeedIncreaser = 5;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
class NormalEnemy : Enemy
{
    public NormalEnemy() {
        Level = Game.rand.Next(Game.AverageEnemyLevel - 5, Game.AverageEnemyLevel
+ 6);
    }
}
class Npc
{

}
class Ogre : NormalEnemy
{
    public Ogre()
    {
        LoadImage("data/images/enemies/normal/ogre.png");

        LifeIncreaser = 33;
        PmIncreaser = 4;
        DamageIncreaser = 16;
        DefenseIncreaser = 8;
        SpeedIncreaser = 2;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;

class Oneiric
{
    [Serializable]
    struct Option
    {
        public byte language;
        public byte difficulty;
        public byte volume;
        public bool fullScreen;
    }

    [Serializable]
    struct GameData
    {
        public int xPlayer;
        public int yPlayer;
        public byte currentDirectionPlayer;
        public int level;
        public int maxiumLife;
        public int actualLife;
        public int lifeIncreaser;
        public int maxiumPm;
        public int actualPm;
        public int pmIncreaser;
        public int damage;
        public int damageIncreaser;
        public int defense;
        public int defenseIncreaser;
        public int speed;
        public int speedIncreaser;
        public int speedPlayer;
        public int lucky;
        public List<Skill> skills;
        public Dictionary<Item,byte> inventory;
        public List<Weapon> weapons;
        public byte actualColMap;
        public byte actualRowMap;
        public int steps;
        public long time;
    }

    public static Dictionary<string, string> ItemsName;
    public static byte Language { get; set; }
    public static byte Difficulty { get; set; }
    public static byte MAX_VOLUME = 10;
    public static byte Volume { get; set; }
    public static bool FullScreen { get; set; }
    public static string[] Languages = { "ESPAÑOL", "ENGLISH" };
    public static string[] Difficultation = { "es", "md", "hr" };
    public static Game g;
    public static OptionsScreen os;

    public static void Inicialize()
    {
        string file = "data/sys/options.dat";
        if (File.Exists(file))
        {
            Option op;
            IFormatter formatter = new BinaryFormatter();
            Stream input = new FileStream(file, FileMode.Open,
                FileAccess.Read, FileShare.Read);
            op = (Option)formatter.Deserialize(input);
            input.Close();

            Language = op.language;
            Difficulty = op.difficulty;
            Volume = op.volume;
            FullScreen = op.fullScreen;
        }
        else
        {
            FullScreen = false;
            Language = 0;
            Difficulty = 1;
            Volume = 10;
        }
        ItemsName = new Dictionary<string, string>();
        LoadItemsName();
    }

    static void Main()
    {
        Inicialize();

        SdlHardware.Init(1200, 768, 24, FullScreen);
        WelcomeScreen w = new WelcomeScreen();
        os = new OptionsScreen();
        LoadGamesScreen lg = new LoadGamesScreen();
        HelpScreen hs = new HelpScreen();

        int option;

        do
        {
            option = w.Run();
            switch (option)
            {
                case 0: break;
                case 1: g = new Game();  g.Run();
                    break;
                case 2:
                    if (lg.Run() == 0)
                    {
                        g.Run();
                    }
                    break;
                case 3: os.Run();
                    SaveOptions();
                    SdlHardware.Init(1200, 768, 24, FullScreen);
                    break;
                case 4: hs.Run();
                    break;
            }
        } while (option != 5);
```

```
}

public static void SaveLog(string error)
{
    StreamWriter file = File.AppendText("data/sys/err.log");

    file.WriteLine(DateTime.Now + " → " + error);

    file.Close();
}

public static void SaveGame(string file)
{
    GameData gD = new GameData();
    gD.xPlayer = g.Mcharacter.GetX();
    gD.yPlayer = g.Mcharacter.GetY();
    gD.currentDirectionPlayer = g.Mcharacter.GetCurrentDirection();
    gD.level = g.Mcharacter.Level;
    gD.maxiumLife = g.Mcharacter.MaxiumLife;
    gD.actualLife = g.Mcharacter.ActualLife;
    gD.lifeIncreaser = g.Mcharacter.LifeIncreaser;
    gD.maxiumPm = g.Mcharacter.MaxiumPm;
    gD.actualPm = g.Mcharacter.ActualPm;
    gD.pmIncreaser = g.Mcharacter.PmIncreaser;
    gD.damage = g.Mcharacter.Damage;
    gD.damageIncreaser = g.Mcharacter.DamageIncreaser;
    gD.defense = g.Mcharacter.Defense;
    gD.defenseIncreaser = g.Mcharacter.DefenseIncreaser;
    gD.speed = g.Mcharacter.Speed;
    gD.speedIncreaser = g.Mcharacter.SpeedIncreaser;
    gD.speedPlayer = g.Mcharacter.GetSpeedX();
    gD.lucky = g.Mcharacter.Lucky;
    gD.skills = g.Mcharacter.Skills;
    gD.inventory = g.Mcharacter.Inventory;
    gD.weapons = g.Mcharacter.Weapons;
    gD.actualColMap = g.Groom.ActualCol;
    gD.actualRowMap = g.Groom.ActualRow;
    gD.steps = g.Steps;
    gD.time = g.Time;

    IFormatter formatter = new BinaryFormatter();
    Stream output = new FileStream(file, FileMode.Create,
        FileAccess.Write, FileShare.None);
    formatter.Serialize(output, gD);
    output.Close();
}

public static void SaveOptions()
{
    Option op = new Option();
    op.language = Language;
    op.difficulty = Difficulty;
    op.volume = Volume;
    op.fullScreen = FullScreen;

    string file = "data/sys/options.dat";
    IFormatter formatter = new BinaryFormatter();
    Stream output = new FileStream(file, FileMode.Create,
        FileAccess.Write, FileShare.None);
    formatter.Serialize(output, op);

    output.Close();
}

public static void LoadGame(string file)
{
    g = new Game();
    GameData gD = new GameData();
    IFormatter formatter = new BinaryFormatter();
    Stream input = new FileStream(file, FileMode.Open,
        FileAccess.Read, FileShare.Read);
    gD = (GameData)formatter.Deserialize(input);
    input.Close();

    g.Mcharacter.MoveTo(gD.xPlayer,gD.yPlayer);
    g.Mcharacter.ChangeDirection(gD.currentDirectionPlayer);
    g.Mcharacter.Level = gD.level;
    g.Mcharacter.MaxiumLife = gD.maxiumLife;
    g.Mcharacter.ActualLife = gD.actualLife;
    g.Mcharacter.LifeIncreaser = gD.lifeIncreaser;
    g.Mcharacter.MaxiumPm = gD.maxiumPm;
    g.Mcharacter.ActualPm = gD.actualPm;
    g.Mcharacter.PmIncreaser = gD.pmIncreaser;
    g.Mcharacter.Damage = gD.damage;
    g.Mcharacter.DamageIncreaser = gD.damageIncreaser;
    g.Mcharacter.Defense = gD.defense;
    g.Mcharacter.DefenseIncreaser = gD.defenseIncreaser;
    g.Mcharacter.Speed = gD.speed;
    g.Mcharacter.SpeedIncreaser = gD.speedIncreaser;
    g.Mcharacter.SetSpeed(gD.speedPlayer,gD.speedPlayer);
    g.Mcharacter.Lucky = gD.lucky;
    g.Mcharacter.Skills = gD.skills;
    g.Mcharacter.Inventory = gD.inventory;
    g.Mcharacter.Weapons = gD.weapons;
    g.Groom.ActualCol = gD.actualColMap;
    g.Groom.ActualRow = gD.actualRowMap;
    g.Groom.UpdateScreenMap(gD.actualColMap,gD.actualRowMap);
    g.Steps = gD.steps;
    g.Time = gD.time;
}
```

```
public static long GetTime(string file)
{
   g = new Game();
   GameData gD = new GameData();
   IFormatter formatter = new BinaryFormatter();
   Stream input = new FileStream(file, FileMode.Open,
      FileAccess.Read, FileShare.Read);
   gD = (GameData)formatter.Deserialize(input);
   input.Close();

   return gD.time;
}

public static void LoadItemsName()
{
   ItemsName.Clear();
   try
   {
      StreamReader file = File.OpenText("data/langs/" +
         Languages[Language].Substring(0, 2).ToLower() +
         "/systemText/items_name.dat");

      string line;

      do
      {
         line = file.ReadLine();

         if (line != null)
         {
            ItemsName.Add((line.Split(';'))[0], (line.Split(';'))[1]);
         }
      } while (line != null);
      file.Close();
   }
   catch (PathTooLongException)
   {
      SaveLog("Path too long Error");
   }
   catch (FileNotFoundException)
   {
      SaveLog("File Not Found");
   }
   catch (IOException e)
   {
      SaveLog("IO Error: " + e);
   }
   catch (Exception e)
   {
```

```
            SaveLog("Error: " + e);
        }
    }
}
class OptionsScreen : Screen
{
    protected Image selector;
    protected int option;
    protected Font font72;

    const int YCURSOR_MAX = 4;
    const int YCURSOR_MIN = 0;

    public OptionsScreen()
        : base(new Image("data/images/other/welcome.jpg"),
            new Font("data/fonts/Joystix.ttf", 28))
    {
        option = 0;
        selector = new Image("data/images/other/selector.png");
        font72 = new Font("data/fonts/Joystix.ttf", 72);
    }

    public int GetChosenOption()
    {
        return option;
    }

    public int Run()
    {
        option = 0;
        LoadText(Oneiric.Languages[Oneiric.Language],"optionMenu");

        SdlHardware.Pause(100);
        do
        {
            SdlHardware.ClearScreen();
            DrawMenu(Oneiric.Languages[Oneiric.Language],
                Oneiric.Difficultation[Oneiric.Difficulty]);
            SdlHardware.ShowHiddenScreen();
            if (SdlHardware.KeyPressed(SdlHardware.KEY_W) && option >
                YCURSOR_MIN)
            {
                option--;
            }
            else if (SdlHardware.KeyPressed(SdlHardware.KEY_S) && option <
                YCURSOR_MAX)
            {
                option++;
            }
            else if (SdlHardware.KeyPressed(SdlHardware.KEY_ESC))
        {
            option = YCURSOR_MAX;
        }
        else if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
        {
            if (option == YCURSOR_MAX)
            {
                Oneiric.LoadItemsName();
                return option;
            }
        }
        else if(SdlHardware.KeyPressed(SdlHardware.KEY_A))
        {
            ChangeOptions(-1, Oneiric.Languages.Length-1,
                Oneiric.Difficultation.Length-1);
        }
        else if (SdlHardware.KeyPressed(SdlHardware.KEY_D))
        {
            ChangeOptions(1, Oneiric.Languages.Length - 1,
                Oneiric.Difficultation.Length - 1);
        }
        SdlHardware.Pause(100);
    }
    while (true);
    //The loop ends when an option is choosed.
}

public void ChangeOptions(sbyte num, int totalLanguages,
    int totalDifficulty)
{
    switch (option)
    {
        case 0:
            if ((Oneiric.Language + num) >= 0 &&
                (Oneiric.Language + num) <= totalLanguages)
            {
                Oneiric.Language += (byte)num;
            }
            break;
        case 1:
            if ((Oneiric.Difficulty + num) >= 0 &&
                (Oneiric.Difficulty + num) <= totalDifficulty)
            {
                Oneiric.Difficulty += (byte)num;
            }
            break;
        case 2:
            Oneiric.FullScreen = Oneiric.FullScreen ? false : true;
            break;
        case 3:
            if ((Oneiric.Volume + num) >= 0 && (Oneiric.Volume + num) <=
                Oneiric.MAX_VOLUME)
            {
                Oneiric.Volume += (byte)num;
            }
            break;
    }
}

public void DrawMenu(string lang, string difficulty)
{
    SdlHardware.DrawHiddenImage(Wallpaper, 0, 0);
    SdlHardware.WriteHiddenText(texts["op"],
       422, 202,
       0x00, 0x00, 0x00,
       font72);
    SdlHardware.WriteHiddenText(texts["op"],
        420, 200,
        0xFF, 0xFF, 0xFF,
        font72);
    SdlHardware.WriteHiddenText(texts["lg"] + ": < " + lang + " >",
       422, 402,
       0x00, 0x00, 0x00,
       Font28);
    SdlHardware.WriteHiddenText(texts["lg"] + ": < " + lang + " >",
        420, 400,
        0xFF, 0xFF, 0xFF,
        Font28);

    SdlHardware.WriteHiddenText(texts["df"] + ": < " + texts[difficulty] +
        " >",
        422, 442,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["df"] + ": < " + texts[difficulty] +
        " >",
        420, 440,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["fs"] + ": < " +
        (Oneiric.FullScreen ? texts["ys"]: texts["no"]) + " >",
        422, 482,
        0x00, 0x00, 0x00,
        Font28);
    SdlHardware.WriteHiddenText(texts["fs"] + ": < " +
        (Oneiric.FullScreen ? texts["ys"] : texts["no"]) + " >",
        420, 480,
        0xFF, 0xFF, 0xFF,
        Font28);
    SdlHardware.WriteHiddenText(texts["vl"] + ": - " +
            new string('!', Oneiric.MAX_VOLUME),
            422, 522,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(texts["vl"] + ": - " +
            new string('!', Oneiric.Volume),
            420, 520,
            0xFF, 0xFF, 0xFF,
            Font28);
        SdlHardware.WriteHiddenText("+",
            900, 522,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText("+",
            900, 520,
            0xFF, 0xFF, 0xFF,
            Font28);
        SdlHardware.WriteHiddenText(texts["bc"],
            422, 562,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(texts["bc"],
            420, 560,
            0xFF, 0xFF, 0xFF,
            Font28);
        SdlHardware.DrawHiddenImage(selector, 320, 380 + 40 * option);
    }
}
class Orc : NormalEnemy
{
    public Orc()
    {
        LoadImage("data/images/enemies/normal/orc.png");

        LifeIncreaser = 33;
        PmIncreaser = 4;
        DamageIncreaser = 8;
        DefenseIncreaser = 16;
        SpeedIncreaser = 2;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
class PassiveSkill : Skill
```

```
{

}
class Puppet : NormalEnemy
{
    public Puppet()
    {
        LoadImage("data/images/enemies/normal/puppet.png");

        LifeIncreaser = 22;
        PmIncreaser = 32;
        DamageIncreaser = 13;
        DefenseIncreaser = 6;
        SpeedIncreaser = 5;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
using System.Collections.Generic;
using System.IO;

class Room
{
    const int TOTAL_IMAGES = 20;
    protected Image[] images = new Image[TOTAL_IMAGES];

    public int mapHeight = 16, mapWidth = 25;
    protected int tileWidth = 48, tileHeight = 48;
    protected int leftMargin = 0, topMargin = 0;

    public int MaxRight = 1150;
    public int MaxDown = 720;

    public string[] levelData;
    protected char[,] mapData;

    public byte ActualCol = 0;
    public byte ActualRow = 0;

    public List<Chest> chests;

    public Room()
    {
        images[0] = new Image("data/images/map/floor1.jpg");
        images[1] = new Image("data/images/map/wall1Down.jpg");
        images[2] = new Image("data/images/map/table.png");
        images[3] = new Image("data/images/map/wall1Center.jpg");
        images[4] = new Image("data/images/map/wall1Up.jpg");
        images[5] = new Image("data/images/map/wall1Center.jpg");
        images[6] = new Image("data/images/map/wall1EndDownCenter.jpg");
        images[7] = new Image("data/images/map/wall1EndUpCenter.jpg");
        images[8] = new Image("data/images/map/wall1MidCenterVertical.jpg");
        images[9] = new Image("data/images/map/wall1MidCenterHorizontal.jpg");
        images[10] = new Image("data/images/map/wall1EndUpRightCorner.jpg");
        images[11] = new Image("data/images/map/wall1EndUpLeftCorner.jpg");
        images[12] = new Image("data/images/map/wall1EndDownRightCorner.jpg");
        images[13] = new Image("data/images/map/wall1EndDownLeftCorner.jpg");
        chests = new List<Chest>();
        LoadDefaultMap();
        mapData = new char[mapWidth, mapHeight];
        UpdateScreenMap(ActualCol, ActualRow);
    }

    public int GetTopMargin() { return topMargin; }

    public void UpdateScreenMap(int col, int row)
    {
        chests.Clear();
        int startRow = row * mapHeight;
        int startCol = col * mapWidth;

        for (int r = 0; r < mapHeight; r++)
        {
            for (int c = 0; c < mapWidth; c++)
            {
                mapData[c, r] = levelData[startRow + r][startCol + c];
            }
        }
    }

    public void DrawOnHiddenScreen()
    {
        for (int row = 0; row < mapHeight; row++)
        {
            for (int col = 0; col < mapWidth; col++)
            {
                int posX = col * tileWidth + leftMargin;
                int posY = row * tileHeight + topMargin;
                switch (mapData[col, row])
                {
                    case 'A': SdlHardware.DrawHiddenImage(images[0], posX,
                        posY); break;
                    case 'W':
                        SdlHardware.DrawHiddenImage(images[1], posX,
                        posY); break;
                    case 'R':
                        SdlHardware.DrawHiddenImage(images[4], posX,
                        posY); break;
                    case 'P':
                        SdlHardware.DrawHiddenImage(images[5], posX,
                        posY); break;
                    case 'I':
                        SdlHardware.DrawHiddenImage(images[11], posX,
                        posY); break;
                    case 'D':
                        SdlHardware.DrawHiddenImage(images[10], posX,
                        posY); break;
                    case 'H':
                        SdlHardware.DrawHiddenImage(images[9], posX,
                        posY); break;
                    case 'V':
                        SdlHardware.DrawHiddenImage(images[8], posX,
                        posY); break;
                    case 'L':
                        SdlHardware.DrawHiddenImage(images[13], posX,
                        posY); break;
                    case 'J':
                        SdlHardware.DrawHiddenImage(images[12], posX,
                        posY); break;
                    case 'C':
                        mapData[col, row] = 'T';
                        SdlHardware.DrawHiddenImage(images[0], posX,
                        posY);
                        SdlHardware.DrawHiddenImage(images[2], posX,
                        posY);
                        Chest c = new Chest("data/images/map/computerOff.png", 1);
                        c.MoveTo(posX,posY);
                        chests.Add(c);
                        break;
                    case 'T':
                        SdlHardware.DrawHiddenImage(images[0], posX,
                        posY);
                        SdlHardware.DrawHiddenImage(images[2], posX,
                        posY);
                        break;
                }
            }
        }
        foreach (Chest c in chests)
            c.DrawOnHiddenScreen();
    }

    public void LoadMap(string route)
    {
        levelData = File.ReadAllLines("data/" + route);
    }

    public void LoadDefaultMap()
    {
        LoadMap("maps/level.map");
    }

    public bool CanMoveTo(int x1, int y1, int x2, int y2)
    {
        for (int column = 0; column < mapWidth; column++)
        {
            for (int row = 0; row < mapHeight; row++)
            {
                char tile = mapData[column, row];

                if (tile != 'A')
                {
                    int x1tile = leftMargin + column * tileWidth;
                    int y1tile = topMargin + row * tileHeight;
                    int x2tile = x1tile + tileWidth;
                    int y2tile = y1tile + tileHeight;

                    if ((x1tile < x2) &&
                        (x2tile > x1) &&
                        (y1tile < y2) &&
                        (y2tile > y1))
                    {
                        return false;
                    }
                }
            }
        }
        return true;
    }
}
using System;
using System.Collections.Generic;
using System.IO;

[Serializable]
abstract class Screen
{
    public Image Wallpaper { get; set; }
    public Font Font28 { get; set; }
    protected Dictionary<string, string> texts;

    public Screen(Image wallpaper, Font font28)
    {
        Wallpaper = wallpaper;
        Font28 = font28;
        texts = new Dictionary<string, string>();
    }

    public void LoadText(string language, string fileName)
    {
        texts.Clear();
        try
        {
            StreamReader file = File.OpenText("data/langs/" +
                language.Substring(0, 2).ToLower() + "/systemText/" + fileName +
                ".dat");

            string line;

            do
            {
                line = file.ReadLine();

                if (line != null)
                {
                    texts.Add((line.Split(';'))[0], (line.Split(';'))[1]);
                }
            } while (line != null);
            file.Close();
        }
        catch (PathTooLongException)
        {
            Oneiric.SaveLog("Path too long Error");
        }
        catch (FileNotFoundException)
        {
            Oneiric.SaveLog("File Not Found");
        }
        catch (IOException e)
        {
            Oneiric.SaveLog("IO Error: " + e);
        }
        catch (Exception e)
        {
            Oneiric.SaveLog("Error: " + e);
        }
    }
}
using System.IO;
using System.Threading;
using Tao.Sdl;
using System;

class SdlHardware
```

```
{
    static IntPtr hiddenScreen;
    static short width, height;

    static short startX, startY; // For Scroll

    static bool isThereJoystick;
    static IntPtr joystick;

    static int mouseClickLapse;
    static int lastMouseClick;


    public static void Init(short w, short h, int colors, bool fullScreen)
    {
        width = w;
        height = h;

        int flags = Sdl.SDL_HWSURFACE | Sdl.SDL_DOUBLEBUF | Sdl.SDL_ANYFORMAT;
        if (fullScreen)
            flags |= Sdl.SDL_FULLSCREEN;
        Sdl.SDL_Init(Sdl.SDL_INIT_EVERYTHING);
        hiddenScreen = Sdl.SDL_SetVideoMode(
            width,
            height,
            colors,
            flags);

        Sdl.SDL_Rect rect2 =
            new Sdl.SDL_Rect(0, 0, (short)width, (short)height);
        Sdl.SDL_SetClipRect(hiddenScreen, ref rect2);

        SdlTtf.TTF_Init();

        // Joystick initialization
        isThereJoystick = true;
        if (Sdl.SDL_NumJoysticks() < 1)
            isThereJoystick = false;

        if (isThereJoystick)
        {
            joystick = Sdl.SDL_JoystickOpen(0);
            if (joystick == IntPtr.Zero)
                isThereJoystick = false;
        }

        // Time lapse between two consecutive mouse clicks,
        // so that they are not too near
        mouseClickLapse = 10;
        lastMouseClick = Sdl.SDL_GetTicks();
```

```
}

public static void ClearScreen()
{
   Sdl.SDL_Rect origin = new Sdl.SDL_Rect(0, 0, width, height);
   Sdl.SDL_FillRect(hiddenScreen, ref origin, 0);
}

public static void DrawHiddenImage(Image image, int x, int y)
{
   drawHiddenImage(image.GetPointer(), x + startX, y + startY);
}

public static void ShowHiddenScreen()
{
   Sdl.SDL_Flip(hiddenScreen);
}

public static bool KeyPressed(int c)
{
   bool pressed = false;
   Sdl.SDL_PumpEvents();
   Sdl.SDL_Event myEvent;
   Sdl.SDL_PollEvent(out myEvent);
   int numkeys;
   byte[] keys = Tao.Sdl.Sdl.SDL_GetKeyState(out numkeys);
   if (keys[c] == 1)
      pressed = true;
   return pressed;
}

public static void Pause(int milisegundos)
{
   Thread.Sleep(milisegundos);
}

public static int GetWidth()
{
   return width;
}

public static int GetHeight()
{
   return height;
}

public static void FatalError(string text)
{
   StreamWriter sw = File.AppendText("errors.log");
   sw.WriteLine(text);
```

```
    sw.Close();
    Console.WriteLine(text);
    Environment.Exit(1);
}

public static void WriteHiddenText(string txt,
    short x, short y, byte r, byte g, byte b, Font f)
{
    Sdl.SDL_Color color = new Sdl.SDL_Color(r, g, b);
    IntPtr textoComoImagen = SdlTtf.TTF_RenderText_Solid(
        f.GetPointer(), txt, color);
    if (textoComoImagen == IntPtr.Zero)
        Environment.Exit(5);

    Sdl.SDL_Rect origen = new Sdl.SDL_Rect(0, 0, width, height);
    Sdl.SDL_Rect dest = new Sdl.SDL_Rect(
        (short)(x + startX), (short)(y + startY),
        width, height);

    Sdl.SDL_BlitSurface(textoComoImagen, ref origen,
        hiddenScreen, ref dest);
}

// Scroll Methods

public static void ResetScroll()
{
    startX = startY = 0;
}

public static void ScrollTo(short newStartX, short newStartY)
{
    startX = newStartX;
    startY = newStartY;
}

public static void ScrollHorizontally(short xDespl)
{
    startX += xDespl;
}

public static void ScrollVertically(short yDespl)
{
    startY += yDespl;
}

// Joystick methods

/** JoystickPressed: returns TRUE if
    *  a certain button in the joystick/gamepad
```

```
   *  has been pressed
   */
public static bool JoystickPressed(int boton)
{
   if (!isThereJoystick)
      return false;

   if (Sdl.SDL_JoystickGetButton(joystick, boton) > 0)
      return true;
   else
      return false;
}

/** JoystickMoved: returns TRUE if
   *  the joystick/gamepad has been moved
   *  up to the limit in any direction
   *  Then, int returns the corresponding
   *  X (1=right, -1=left)
   *  and Y (1=down, -1=up)
   */
public static bool JoystickMoved(out int posX, out int posY)
{
   posX = 0; posY = 0;
   if (!isThereJoystick)
      return false;

   posX = Sdl.SDL_JoystickGetAxis(joystick, 0);  // Leo valores (hasta 32768)
   posY = Sdl.SDL_JoystickGetAxis(joystick, 1);
   // Normalizo valores
   if (posX == -32768) posX = -1;  // Normalizo, a -1, +1 o 0
   else if (posX == 32767) posX = 1;
   else posX = 0;
   if (posY == -32768) posY = -1;
   else if (posY == 32767) posY = 1;
   else posY = 0;

   if ((posX != 0) || (posY != 0))
      return true;
   else
      return false;
}


/** JoystickMovedRight: returns TRUE if
   *  the joystick/gamepad has been moved
   *  completely to the right
   */
public static bool JoystickMovedRight()
{
   if (!isThereJoystick)
```

```
        return false;

    int posX = 0, posY = 0;
    if (JoystickMoved(out posX, out posY) && (posX == 1))
        return true;
    else
        return false;
}

/** JoystickMovedLeft: returns TRUE if
    *  the joystick/gamepad has been moved
    *  completely to the left
    */
public static bool JoystickMovedLeft()
{
    if (!isThereJoystick)
        return false;

    int posX = 0, posY = 0;
    if (JoystickMoved(out posX, out posY) && (posX == -1))
        return true;
    else
        return false;
}


/** JoystickMovedUp: returns TRUE if
    *  the joystick/gamepad has been moved
    *  completely upwards
    */
public static bool JoystickMovedUp()
{
    if (!isThereJoystick)
        return false;

    int posX = 0, posY = 0;
    if (JoystickMoved(out posX, out posY) && (posY == -1))
        return true;
    else
        return false;
}


/** JoystickMovedDown: returns TRUE if
    *  the joystick/gamepad has been moved
    *  completely downwards
    */
public static bool JoystickMovedDown()
{
    if (!isThereJoystick)
```

```
        return false;

    int posX = 0, posY = 0;
    if (JoystickMoved(out posX, out posY) && (posY == 1))
        return true;
    else
        return false;
}


/** GetMouseX: returns the current X
    *  coordinate of the mouse position
    */
public static int GetMouseX()
{
    int posX = 0, posY = 0;
    Sdl.SDL_PumpEvents();
    Sdl.SDL_GetMouseState(out posX, out posY);
    return posX;
}


/** GetMouseY: returns the current Y
    *  coordinate of the mouse position
    */
public static int GetMouseY()
{
    int posX = 0, posY = 0;
    Sdl.SDL_PumpEvents();
    Sdl.SDL_GetMouseState(out posX, out posY);
    return posY;
}


/** MouseClicked: return TRUE if
    *  the (left) mouse button has been clicked
    */
public static bool MouseClicked()
{
    int posX = 0, posY = 0;

    Sdl.SDL_PumpEvents();

    // To avoid two consecutive clicks
    int now = Sdl.SDL_GetTicks();
    if (now - lastMouseClick < mouseClickLapse)
        return false;

    // Ahora miro si realmente hay pulsación
    if ((Sdl.SDL_GetMouseState(out posX, out posY) & Sdl.SDL_BUTTON(1)) == 1)
    {
        lastMouseClick = now;
        return true;
    }
    else
        return false;
}


// Private (auxiliar) methods

private static void drawHiddenImage(IntPtr image, int x, int y)
{
    Sdl.SDL_Rect origin = new Sdl.SDL_Rect(0, 0, width, height);
    Sdl.SDL_Rect dest = new Sdl.SDL_Rect((short)x, (short)y,
        width, height);
    Sdl.SDL_BlitSurface(image, ref origin, hiddenScreen, ref dest);
}


// Alternate key definitions

public static int KEY_ESC = Sdl.SDLK_ESCAPE;
public static int KEY_SPC = Sdl.SDLK_SPACE;
public static int KEY_A = Sdl.SDLK_a;
public static int KEY_B = Sdl.SDLK_b;
public static int KEY_C = Sdl.SDLK_c;
public static int KEY_D = Sdl.SDLK_d;
public static int KEY_E = Sdl.SDLK_e;
public static int KEY_F = Sdl.SDLK_f;
public static int KEY_G = Sdl.SDLK_g;
public static int KEY_H = Sdl.SDLK_h;
public static int KEY_I = Sdl.SDLK_i;
public static int KEY_J = Sdl.SDLK_j;
public static int KEY_K = Sdl.SDLK_k;
public static int KEY_L = Sdl.SDLK_l;
public static int KEY_M = Sdl.SDLK_m;
public static int KEY_N = Sdl.SDLK_n;
public static int KEY_O = Sdl.SDLK_o;
public static int KEY_P = Sdl.SDLK_p;
public static int KEY_Q = Sdl.SDLK_q;
public static int KEY_R = Sdl.SDLK_r;
public static int KEY_S = Sdl.SDLK_s;
public static int KEY_T = Sdl.SDLK_t;
public static int KEY_U = Sdl.SDLK_u;
public static int KEY_V = Sdl.SDLK_v;
public static int KEY_W = Sdl.SDLK_w;
public static int KEY_X = Sdl.SDLK_x;
public static int KEY_Y = Sdl.SDLK_y;
public static int KEY_Z = Sdl.SDLK_z;
```

```
    public static int KEY_1 = Sdl.SDLK_1;
    public static int KEY_2 = Sdl.SDLK_2;
    public static int KEY_3 = Sdl.SDLK_3;
    public static int KEY_4 = Sdl.SDLK_4;
    public static int KEY_5 = Sdl.SDLK_5;
    public static int KEY_6 = Sdl.SDLK_6;
    public static int KEY_7 = Sdl.SDLK_7;
    public static int KEY_8 = Sdl.SDLK_8;
    public static int KEY_9 = Sdl.SDLK_9;
    public static int KEY_0 = Sdl.SDLK_0;
    public static int KEY_UP = Sdl.SDLK_UP;
    public static int KEY_DOWN = Sdl.SDLK_DOWN;
    public static int KEY_RIGHT = Sdl.SDLK_RIGHT;
    public static int KEY_LEFT = Sdl.SDLK_LEFT;
    public static int KEY_RETURN = Sdl.SDLK_RETURN;
}
class Skeleton : NormalEnemy
{
    public Skeleton()
    {
        LoadImage("data/images/enemies/normal/skeleton.png");

        LifeIncreaser = 19;
        PmIncreaser = 29;
        DamageIncreaser = 15;
        DefenseIncreaser = 5;
        SpeedIncreaser = 6;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
class Skill
{

}
class Sound
{

}
using System;

[Serializable]
class Sprite
{
```

```
protected int x, y;
protected int startX, startY;
protected int width, height;
protected int xSpeed, ySpeed;
protected bool visible;
protected Image image;
protected Image[][] sequence;
protected bool containsSequence;
protected int currentFrame;

protected byte numDirections = 11;
protected byte currentDirection;
public const byte RIGHT = 0;
public const byte LEFT = 1;
public const byte DOWN = 2;
public const byte UP = 3;
public const byte DOWNRIGHT = 4;
public const byte DOWNLEFT = 5;
public const byte UPRIGHT = 6;
public const byte UPLEFT = 7;
public const byte APPEARING = 8;
public const byte DISAPPEARING = 9;
public const byte JUMPING = 9;

public Sprite()
{
    startX = -1;
    startY = -1;
    width = 32;
    height = 32;
    visible = true;
    sequence = new Image[numDirections][];
    currentDirection = RIGHT;
}

public Sprite(string imageName)
    : this()
{
    LoadImage(imageName);
}

public Sprite(string[] imageNames)
    : this()
{
    LoadSequence(imageNames);
}

public void LoadImage(string name)
{
    image = new Image(name);
    containsSequence = false;
}

public void LoadSequence(byte direction, string[] names)
{
    int amountOfFrames = names.Length;
    sequence[direction] = new Image[amountOfFrames];
    for (int i = 0; i < amountOfFrames; i++)
        sequence[direction][i] = new Image(names[i]);
    containsSequence = true;
    currentFrame = 0;
}

public void LoadSequence(string[] names)
{
    LoadSequence(RIGHT, names);
}

public int GetX()
{
    return x;
}

public int GetY()
{
    return y;
}

public int GetWidth()
{
    return width;
}

public int GetHeight()
{
    return height;
}

public int GetSpeedX()
{
    return xSpeed;
}

public int GetSpeedY()
{
    return ySpeed;
}

public bool IsVisible()
{
    return visible;
}

public byte GetCurrentDirection()
{
    return currentDirection;
}

public void MoveTo(int newX, int newY)
{
    x = newX;
    y = newY;
    if (startX == -1)
    {
        startX = x;
        startY = y;
    }
}

public void SetSpeed(int newXSpeed, int newYSpeed)
{
    xSpeed = newXSpeed;
    ySpeed = newYSpeed;
}

public void Show()
{
    visible = true;
}

public void Hide()
{
    visible = false;
}

public virtual void Move()
{
    // To be redefined in subclasses
}

public virtual void DrawOnHiddenScreen()
{
    if (!visible)
        return;

    if (containsSequence)
        SdlHardware.DrawHiddenImage(
            sequence[currentDirection][currentFrame], x, y);
    else
        SdlHardware.DrawHiddenImage(image, x, y);
    }

    public void NextFrame()
    {
        currentFrame++;
        if (currentFrame >= sequence[currentDirection].Length)
            currentFrame = 0;
    }

    public void ChangeDirection(byte newDirection)
    {
        if (!containsSequence) return;
        if (currentDirection != newDirection)
        {
            currentDirection = newDirection;
            currentFrame = 0;
        }

    }

    public bool CollisionsWith(Sprite otherSprite)
    {
        return (visible && otherSprite.IsVisible() &&
            CollisionsWith(otherSprite.GetX(),
                otherSprite.GetY(),
                otherSprite.GetX() + otherSprite.GetWidth(),
                otherSprite.GetY() + otherSprite.GetHeight()));
    }

    public bool CollisionsWith(int xStart, int yStart, int xEnd, int yEnd)
    {
        if (visible &&
                (x < xEnd) &&
                (x + width > xStart) &&
                (y < yEnd) &&
                (y + height > yStart)
                )
            return true;
        return false;
    }

    public void Restart()
    {
        x = startX;
        y = startY;
    }
}
class Succubus : NormalEnemy
{
    public Succubus()
    {
        LoadImage("data/images/enemies/normal/succubus.png");

        LifeIncreaser = 31;
        PmIncreaser = 28;
        DamageIncreaser = 16;
        DefenseIncreaser = 10;
        SpeedIncreaser = 11;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
class Vampire : NormalEnemy
{
    public Vampire()
    {
        LoadImage("data/images/enemies/normal/vampire.png");

        LifeIncreaser = 37;
        PmIncreaser = 25;
        DamageIncreaser = 13;
        DefenseIncreaser = 7;
        SpeedIncreaser = 10;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
using System;

[Serializable]
class Weapon : EquipableItem
{
    public Weapon(string name, int lfI, int pmI, int daI, int deI,
        int spI, int luI, int rarity)
        : base(name, lfI, pmI, daI, deI, spI, luI, rarity)
    {

    }
```

```
}
using System;

[Serializable]
class WearingItem : EquipableItem
{
    public WearingItem(string name, int lfI, int pmI, int daI, int deI,
        int spI, int luI, int rarity)
        : base(name, lfI, pmI, daI, deI, spI, luI, rarity)
    {

    }
}
using System;
using System.Collections.Generic;

class WelcomeScreen : Screen
{
    protected Image selector;
    protected int option;
    protected Font font94;

    const int YCURSOR_MAX = 5;
    const int YCURSOR_MIN = 0;

    string[,] title = new string[,] { { "O", "200" }, { "N", "200" },
        { "E", "200" }, {"I", "200"}, { "R", "200" }, { "I", "200" },
        { "C", "200" } };

    byte actualLetter = 0;
    bool isDown = true;
    const string LETTER_Y_MAX = "250";
    const string LETTER_Y_MIN = "200";
    const int Y_MODIFIER = 25;

    public WelcomeScreen()
        : base(new Image("data/images/other/welcome.jpg"),
            new Font("data/fonts/Joystix.ttf", 28))
    {
        option = 0;
        selector = new Image("data/images/other/selector.png");
        font94 = new Font("data/fonts/Joystix.ttf", 94);
        texts = new Dictionary<string, string>();
    }

    public int GetChosenOption()
    {
        return option;
    }
```

```
public int Run()
{
   option = 0;
   LoadText(Oneiric.Languages[Oneiric.Language], "mainMenu");

   do
   {
      SdlHardware.ClearScreen();
      MoveLetters();
      DrawMenu();
      SdlHardware.ShowHiddenScreen();
      if (SdlHardware.KeyPressed(SdlHardware.KEY_W) && option >
         YCURSOR_MIN)
      {
         option--;
      }
      else if (SdlHardware.KeyPressed(SdlHardware.KEY_S) && option <
         YCURSOR_MAX)
      {
         option++;
      }
      else if (SdlHardware.KeyPressed(SdlHardware.KEY_ESC))
      {
         option = YCURSOR_MAX;
      }
      else if (SdlHardware.KeyPressed(SdlHardware.KEY_RETURN))
      {
         return option;
      }
      SdlHardware.Pause(100);
   }
   while (true);
   //The loop ends when an option is choosed.
}

public void MoveLetters()
{
   if (isDown)
   {
      title[actualLetter, 1] = (Convert.ToInt16(title[actualLetter, 1]) +
         Y_MODIFIER).ToString();
   }
   else
   {
      title[actualLetter, 1] = (Convert.ToInt16(title[actualLetter, 1]) -
         Y_MODIFIER).ToString();
   }

   if (title[actualLetter, 1] == LETTER_Y_MAX && isDown)
   {
```

```
            isDown = false;
        }
        else if (title[actualLetter, 1] == LETTER_Y_MIN && !isDown)
        {
            isDown = true;
            actualLetter++;
        }

        if (actualLetter == title.GetLength(0))
        {
            actualLetter = 0;
        }
    }

    public void DrawMenu()
    {
        SdlHardware.DrawHiddenImage(Wallpaper, 0, 0);
        SdlHardware.WriteHiddenText(title[0,0],
           352, (short)(Convert.ToInt16(title[0, 1]) + 2),
           0x00, 0x00, 0x00,
           font94);
        SdlHardware.WriteHiddenText(title[0, 0],
            350, Convert.ToInt16(title[0, 1]),
            0xFF, 0xFF, 0xFF,
            font94);
        SdlHardware.WriteHiddenText(title[1, 0],
           424, (short)(Convert.ToInt16(title[1, 1]) + 2),
           0x00, 0x00, 0x00,
           font94);
        SdlHardware.WriteHiddenText(title[1, 0],
            422, Convert.ToInt16(title[1, 1]),
            0xFF, 0xFF, 0xFF,
            font94);
        SdlHardware.WriteHiddenText(title[2, 0],
           494, (short)(Convert.ToInt16(title[2, 1]) + 2),
           0x00, 0x00, 0x00,
           font94);
        SdlHardware.WriteHiddenText(title[2, 0],
            492, Convert.ToInt16(title[2, 1]),
            0xFF, 0xFF, 0xFF,
            font94);
        SdlHardware.WriteHiddenText(title[3, 0],
           566, (short)(Convert.ToInt16(title[3, 1]) + 2),
           0x00, 0x00, 0x00,
           font94);
        SdlHardware.WriteHiddenText(title[3, 0],
            564, Convert.ToInt16(title[3, 1]),
            0xFF, 0xFF, 0xFF,
            font94);
        SdlHardware.WriteHiddenText(title[4, 0],
   638, (short)(Convert.ToInt16(title[4, 1]) + 2),
   0x00, 0x00, 0x00,
   font94);
SdlHardware.WriteHiddenText(title[4, 0],
    636, Convert.ToInt16(title[4, 1]),
    0xFF, 0xFF, 0xFF,
    font94);
SdlHardware.WriteHiddenText(title[5, 0],
   710, (short)(Convert.ToInt16(title[5, 1]) + 2),
   0x00, 0x00, 0x00,
   font94);
SdlHardware.WriteHiddenText(title[5, 0],
    708, Convert.ToInt16(title[5, 1]),
    0xFF, 0xFF, 0xFF,
    font94);
SdlHardware.WriteHiddenText(title[6, 0],
   782, (short)(Convert.ToInt16(title[6, 1])+2),
   0x00, 0x00, 0x00,
   font94);
SdlHardware.WriteHiddenText(title[6, 0],
    780, Convert.ToInt16(title[6, 1]),
    0xFF, 0xFF, 0xFF,
    font94);
SdlHardware.WriteHiddenText(texts["co"],
   422, 482,
   0x00, 0x00, 0x00,
   Font28);
SdlHardware.WriteHiddenText(texts["co"],
    420, 480,
    0xFF, 0xFF, 0xFF,
    Font28);
SdlHardware.WriteHiddenText(texts["ng"],
    422, 522,
    0x00, 0x00, 0x00,
    Font28);
SdlHardware.WriteHiddenText(texts["ng"],
    420, 520,
    0xFF, 0xFF, 0xFF,
    Font28);
SdlHardware.WriteHiddenText(texts["lg"],
    422, 562,
    0x00, 0x00, 0x00,
    Font28);
SdlHardware.WriteHiddenText(texts["lg"],
    420, 560,
    0xFF, 0xFF, 0xFF,
    Font28);
SdlHardware.WriteHiddenText(texts["op"],
    422, 602,
    0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(texts["op"],
            420, 600,
            0xFF, 0xFF, 0xFF,
            Font28);
        SdlHardware.WriteHiddenText(texts["hl"],
            422, 642,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(texts["hl"],
            420, 640,
            0xFF, 0xFF, 0xFF,
            Font28);
        SdlHardware.WriteHiddenText(texts["ex"],
            422, 682,
            0x00, 0x00, 0x00,
            Font28);
        SdlHardware.WriteHiddenText(texts["ex"],
            420, 680,
            0xFF, 0xFF, 0xFF,
            Font28);
        SdlHardware.DrawHiddenImage(selector, 320, 460 + 40 * option);
    }
}
class Werewolf : NormalEnemy
{
    public Werewolf()
    {
        LoadImage("data/images/enemies/normal/werewolf.png");

        LifeIncreaser = 37;
        PmIncreaser = 12;
        DamageIncreaser = 19;
        DefenseIncreaser = 9;
        SpeedIncreaser = 13;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
class Zombie : NormalEnemy
{
    public Zombie()
    {
        LoadImage("data/images/enemies/normal/zombie.png");
        LifeIncreaser = 21;
        PmIncreaser = 28;
        DamageIncreaser = 23;
        DefenseIncreaser = 5;
        SpeedIncreaser = 5;

        MaxiumLife = LifeIncreaser * Level;
        MaxiumPm = PmIncreaser * Level;
        Damage = DamageIncreaser * Level;
        Defense = DefenseIncreaser * Level;
        Speed = SpeedIncreaser * Level;

        ActualLife = MaxiumLife;
    }
}
```