|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Título del proyecto** | | Zombieland | | |
| **Fecha de inicio** | | 4/05/2015 | **Fecha de entrega** | 21/05/2015 |
| **Responsables** | | Soulas, Valentín. | | |
| **Descripción corta** | | Zombieland es un shooter-survival en 2D que representa un espacio tridimensional en modo isométrico. El jugador, sobreviviente de un apocalipsis zombie, deberá llegar al objetivo. En el camino sera atacado constantemente por distintos enemigos. A medida que se matan enemigos el jugador obtendrá créditos los cuales pueden ser usados en una **\***tienda para comprar armas, mejoras y municiones. | | |
| **Descripción detallada (alcance)**  **OBJETIVO**  El objetivo principal del juego es llegar a la zona segura.  El jugador aparecerá en unas coordenadas determinadas por el mapa en el que le toque jugar. Una vez dentro el jugador deberá llegar al objetivo específico de ese mapa. Para así entonces pasar al siguiente mapa. El objetivo del último mapa será el objetivo final del juego, el cual es llegar a la zona segura para ser rescatado.  Entre cambios de mapa, la dificultad aumentará. A su vez el jugador tendrá la posibilidad de comprar mejores armas para poder enfrentar estos nuevos enemigos que son aún más poderosos. Esto se logra comprando o mejorando las armas en una **\***“tienda” la cual podrá ser accedida desde el menú del juego.  Una vez que el jugador llegue al objetivo final y gane el juego tendrá la opción de guardar su puntaje para luego aparecer en la tabla de los mejores puntajes.  **\***También se le preguntará si desea jugar el juego nuevamente pero esta vez en una dificultad general aún mayor.  **ESTADOS**  El juego se manejara con estados.  Los estados principales del juego serán: MAINMENU, OPCIONES, INGAMEMENU, STORE, START\_PLAYING, PLAYING, HELP, GAMEOVER, EXIT.  El programa ejecutará SOLO aquellas funciones que corresponda al estado actual del juego.  **RESTRICCIONES**   1. **El Jugador**    1. Habrá un solo jugador    2. Se mueve en 8 direcciones    3. Velocidad de movimiento fija    4. El jugador controla el movimiento del fondo y todo lo que dependa del mismo (fondo)    5. Tendrá la posibilidad de teletransportarse a la ubicación del mouse en la pantalla (Solo para desarrolladores)    6. Colisiona con los bordes y objetos estáticos    7. Aparecerá con un tipo de arma    8. **\***Al matar un enemigo, el jugador obtendrá créditos, los cuales podrá gastar en la “tienda”    9. A medida que obtenga nuevas armas estas se guardan en el inventario 2. **Las Armas**    1. Habrán 4 tipos de armas    2. El jugador empieza siempre con 1 tipo de arma    3. El resto de las armas podrán comprarse en la tienda con los créditos obtenidos al matar los enemigos    4. Las armas ya obtenidas se guardan en el inventario    5. Las armas pueden mejorarse con los creditos    6. Las balas:       1. Viajarán en dirección al mouse en el momento que fueron disparadas       2. Harán un daño preestablecido dependiendo el tipo de arma y sus mejoras       3. **\***El jugador tendrá mejor punterìa al avanzar en el juego. 3. **Municiones**    1. Las municiones no son infinitas    2. El jugador aparecerá con una cantidad limitada de municiones.    3. Las municiones aparecerán en lugares random y además podrán comprarse en la tienda 4. **Mapa**    1. Habrán varios mapas    2. El tamaño será mayor al de la pantalla    3. Cada mapa tendrá un punto de partida y otro de llegada    4. El movimiento del mapa depende directamente del jugador (ver punto 1.4)    5. En el mapa habrán objetos estáticos colisionables       1. El espacio que ocupe la base de un objeto estático será considerado un espacio reservado por el cual nada podrá atravesarlo, ya sea el jugador, un enemigo o las balas 5. **Minimapa**    1. El minimapa estará en la esquina inferior izquierda    2. La escala es 1 a 20    3. Se representarán la ubicación del jugador 6. **Enemigos**    1. La velocidad de movimiento será menor a la del jugador e incrementara acorde al nivel del jugador    2. Podrán moverse en 8 direcciones    3. Estarán repartidos por todo el mapa    4. Atacará solo si el jugador esta en “rango”    5. Si el enemigo está en contacto con el jugador, este recibirá un daño    6. Habrán distintos tipos de enemigos    7. **\*** Habrán enemigos denominados “BOSS” los cuales son mas poderosos y mas difíciles de matar. 7. **Mouse**    1. El mouse se utilizará para dirigir los disparos ¡ 8. **Menu**    1. El menú principal tendrá los siguientes items:       1. “Comenzar partida” Aquí el usuario ingresa su nombre y comienza el juego       2. “Opciones”       3. “Ayuda” Para informar el objetivo del juego y mostrar los controles       4. “Ranking” Se listan los mejores puntajes       5. “Salir” Cierra el juego    2. El menú del juego tendrá los siguientes items:       1. “Continuar” Reanuda el juego donde estaba       2. **\***“Tienda” Para comprar armas, mejoras, etc.       3. “Opciones”       4. “Ayuda” Para informar el objetivo del juego y mostrar los controles       5. “Menu Principal” Se le preguntara al usuario si desea abandonar la partida   **\* Posibles mejoras** | | | | |
| **Archivos a utilizar (nombre y descripción de los campos)**  Highscores.dat // Guardar y levantar los puntajes  Gamesaves.dat // Guardar y levantar las partidas | | | | |
| **Clases a utilizar (nombre, propiedades y prototipo de los métodos)**   |  | | --- | | CLASS ALLEGRO | | private:         double new\_time,old\_time;         int cont,fps,last\_fps; | | public:         Allegro(int,int); // Constructor encargado de iniciar el allegro. Los parametros que recibe son el ancho y alto del juego.         void install\_addons();         void load\_fonts();         void register\_events();         void terminate\_allegro();      // Termina allegro         /// Mostrar Info         void showMousePos(int,int);         void showFps(); |   **CLASE PARA ADMINISTRAR LOS ESTADOS DEL PROGRAMA**   |  | | --- | | CLASS CGameState | | private:     int actual\_state; | | public:     void changeState(int);     int getState(); |   **CLASES DE ESTADO DEL PROGRAMA**   |  | | --- | | CLASS CMainMenu | | private:     int selec; | | public:     void open\_mainmenu();     void render\_mainmenu();     CMainMenu(){         selec = 0;     } |  |  | | --- | | CLASS CIngameMenu | | private:     int selec; | | public:     void open\_ingamemenu();     void render\_ingamemenu();     CIngameMenu(){         selec = 0;     } |  |  | | --- | | CLASS CGame | | private:     int selec; | | public:     void open\_game();     void render\_game();     CGame(){         selec = 0;     } |  |  | | --- | | CLASS CHelp | | private:     int selec; | | public:     void open\_help();     void render\_help();     CHelp(){         selec = 0;     } |   **CLASES DEL JUEGO**   |  | | --- | | CLASS PLAYER | | private:         int x,y;                 // Posicion en la pantalla         int xa,ya;             // Posicion en la pantalla anterior         int xmap,ymap;   // Posicion en el mapa         int w,h;              // Ancho y alto del BITMAP         int direc,frame;    // Para dibujar la parte del BITMAP correspondiente         int vel, hp; // Velocidad y Vida           bool alive;         char name[20];         rectCol rect;       // Region de colision (por lo general mas chica que el BMP)         ALLEGRO\_BITMAP \*bmp; | | public:         void init(int,int,char \*,char \*,int,int,int,int,int,int);         ~player(){}         void draw();                      // Dibujar en pantalla en X,Y         void draw\_minimap();             // Dibujar circulo en minimapa         void showInfo();                // Muestra posiciones por pantalla         void updatePos(int);                  // Cambia la posicion recibe DIREC         void updatePosAnt();              // Actualiza Xant,Yant         void updatePosInMap(int,int);    // Actualiza Xmap,Ymap         void teleport();                // Teletransporta a la ubicacion del mouse         void kill();  int hit\_recived(int);          // Disminuye la vida y devuelve la vida actual         void destroy\_bitmap() {al\_destroy\_bitmap(bmp);}         // Setters && Getters         int getX(){return x;}         int getY(){return y;}         int getXMAP(){return xmap;}         int getYMAP(){return ymap;}         void setAnt(){xa = x; ya = y;} |  |  | | --- | | **CLASS MAPA** | | private:  int x,y;  int w,h;  ALLEGRO\_BITMAP \*bmp,\*bmp\_col,\*bmp\_minimap; | | public:  mapa(int,int,char \*,char \*,char \*);  ~mapa(){}  void updatePos(int);  void draw();  void draw\_minimap();  void delete\_bmp();  // Setters & Getters  int getX(){return x;}  int getY(){return y;}  int getW(){return w;}  int getH(){return h;}  //void colisionPixel(mapa,jugador); |  |  | | --- | | **CLASS ENTORNO** | | private:  int x,y;  int w,h;  ALLEGRO\_BITMAP \*bmp; | | public:  entorno(int ,int , char \*);  ~entorno(){}  void draw(mapa);  void delete\_bmp(){destroy\_bitmap(bmp);} |  |  | | --- | | **CLASS ENEMY** | | private:  int x,y; // Posicion en la pantalla  int xa,ya; // Posicion en la pantalla anterior  int xmap,ymap; // Posicion en el mapa  int w,h; // Ancho y alto del BITMAP  int vel; // Velocidad de movimiento  int direc,frame; // Para dibujar la parte del BITMAP correspondiente  int stamina; // Vida  bool alive;  rectCol rect; | | public:  enemy();  ~enemy(){}  void spawn(int,int,int,int); // Funcion para crear enemigos. Recibe coordenadas del mapa para evitar  // crear un enemigo dentro de la pantalla  void updatePos(int,int); // Mueve al enemigo. Recibe coordenadas del jugador para establecer una direccion  void draw(int,int);  void draw\_minimap();  bool getAlive(){return alive;}  void setX(int X){x = X;}  void setY(int Y){y = Y;}  int getX(){return x;}  int getY(){return y;}  int getXMAP(){return xmap;}  int getYMAP(){return ymap;} |  |  | | --- | | **CLASS BULLET** | | private:  float x,y;  int w,h;  float vx,vy;  double rad;  bool shooting; | | public:  void shoot(int,int);  void updateBullet();  void drawBullet();  bool getShooting(){return shooting;}  bullet(){  shooting = false;  } |  |  | | --- | | **CLASS RANDOM** | | No hay parte privada ya que los métodos son estaticos y nunca se instancia un objeto de la clase | | public:  static int inicializar(time\_t seed=time(NULL)){  srand(seed);  }  static int getRandom(int limite){  return rand() % limite;  }  static int getRandom(int inicio, int fin){  return rand() % fin + inicio;  } |  |  | | --- | | **CLASS COLLISION** | | No hay parte privada ya que los métodos son estaticos y nunca se instancia un objeto de la clase | | public:  static bool screenBorder(rectCol,int,int);  static bool boundingBox(rectCol,int,int,rectCol,int,int);  static bool pixel(rectCol);  static bool circle\_pixel(int,int,int,int,int); | | | | | |
| **Corrección** | | | | |
| **Fecha** | **Estado** | **Observaciones** | | **Firma** |
|  |  |  | |  |
|  |  |  | |  |