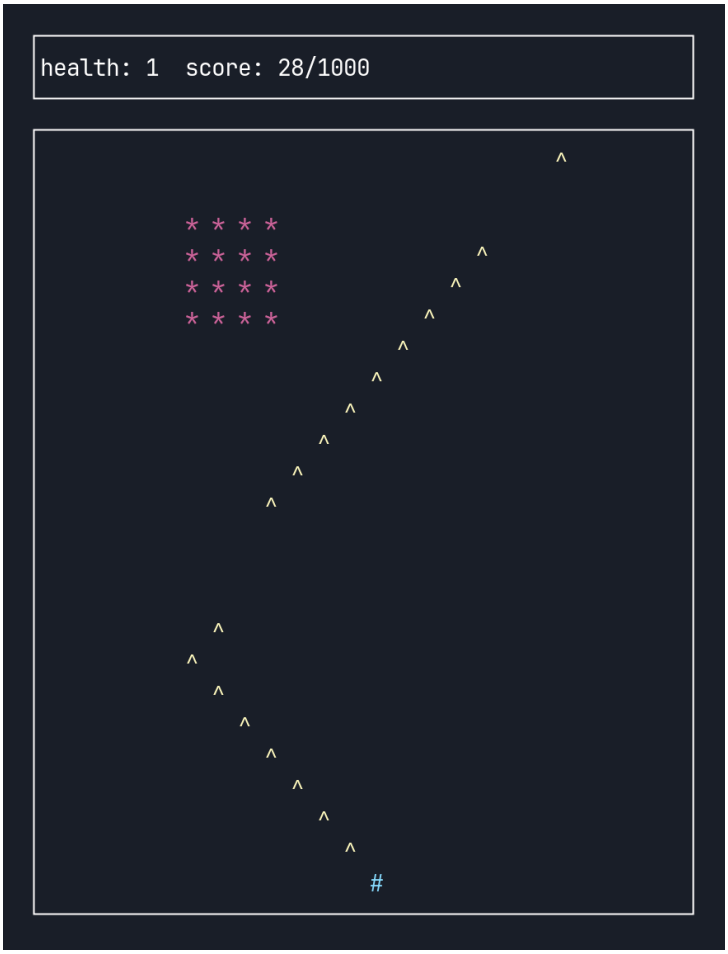

جنگ ستارگان

1402/12/07

—

امیرعباس فاضلی نیا

شماره دانشجویی: 40212358031

دانشگاه بوعلی سینا، همدان

گیتهاب: github.com/utilyre/starwars-final

معرفی بازی

گیم پلی

بازیکن یک سفینه فضایی در پایین صفحه را کنترل می کند که می تواند به چپ و راست حرکت کرده و در هر حرکت، به سمت سفینه های دشمن شلیک کند. این دشمن ها در هر حرکت سفینه خودی، یک خانه نزدیک تر می شوند و اگر تا قبل از رسیدن به بازیکن نابود نشوند، بازیکن یکی از جان های خود را از دست می دهد.

اگر جان های شما تمام شوند، شما خواهید باخت و باید بازی را از سر بگیرید. اما اگر با از بین بردن سفینه های دشمن امتیاز شما به حد نصاب تعیین شده در ابتدای بازی برسد، برنده می شوید و حتی می توانید بازی را ادامه دهید.

دشمن ها

1. سفینه **Dart**: تعداد جان های این سفینه برابر **1** و سایز آن **1×1** است.
2. سفینه **Striker**: تعداد جان های این سفینه برابر **2** و سایز آن **2×2** است.
3. سفینه **Wraith**: تعداد جان های این سفینه برابر **4** و سایز آن **3×3** است.
4. سفینه **Banshee**: تعداد جان های این سفینه برابر **6** و سایز آن **4×4** است.

ساختار پروژه

پروژه متشکل از 3 استراکت و 5 کلاس می باشد.

استراکت Vec2

یک بردار دو بعدی را نمایندگی می کند و برای ذخیره موقعیت موجودیت های بازی به کار می آید.

دارای 2 فیلد *x* و *y* است که به ترتیب فاصله نقطه از محور y ها و محور x ها را نشان می دهند.

دو نسخه constructor دارد؛ یکی بردار پیشفرض (0, 0) را می سازد و دیگری با توجه به پارامتر های داده شده، *x* و *y*، بردار را ایجاد می کند.

متد *add* اجزای یک **Vec2** دیگر را بصورت متناظر به اجزای خود اضافه می کند و حاصل را به صورت یک **Vec2** جدید خروجی می دهد.

متد *clamp* اجزای بردار را بین دو مقدار *min* و *max* که به صورت پارامتر داده شده محدود می کند و حاصل را به صورت یک **Vec2** جدید خروجی می دهد.

```
/**
 * Represents a vector on two-dimensional plane.
 */
struct Vec2
{
    int y, x;

    Vec2();
    Vec2(int y, int x);

    /**
     * Adds `other` to the vector.
     */
    Vec2 add(Vec2 other);

    /**
     * Clamps the vector between `min` and `max`.
     */
    Vec2 clamp(Vec2 min, Vec2 max);
};
```

استراکت Rect

یک مستطیل را نمایندگی می کند و عموما برای ذخیره فضای برخورد موجودیت ها به کار می رود.

دارای 2 فیلد *size* و *translation* است که به ترتیب اندازه و موقعیت گوشه چپ بالای مستطیل را نشان می دهند.

```
/**
 * Size of the rectangle.
 */
Vec2 size;

/**
 * Translation of the upper-left corner of the rectangle.
 */
Vec2 translation;
```

سه نسخه constructor دارد؛ اولی از constructor پیشفرض **Vec2** استفاده می کند، دومی مقدار فیلد های *size* و *translation* را بطور مستقیم دریافت و تنظیم می کند و سومی هم نسخه بسط داده شده دومی می باشد.

متد های *top* و *bottom* به ترتیب موقعیت *y* ضلع های بالا و پایین را خروجی می دهند.

متد های *left* و *right* به ترتیب موقعیت *x* ضلع های چپ و راست را خروجی می دهند.

متد *collides_with* چک می کند که آیا این مستطیل با مستطیل *rect* فصل مشترک دارد یا خیر.

```
/**
 * Returns the y-axis translation of the top edge.
 */
int top() const;

/**
 * Returns the x-axis translation of the left edge.
 */
int left() const;

/**
 * Returns the y-axis translation of the bottom edge.
 */
int bottom() const;

/**
 * Returns the x-axis translation of the right edge.
 */
int right() const;

/**
 * Checks whether this rectangle collides with `rect`.
 */
bool collides_with(Rect rect) const;
```

اگر چهار حالت زیر همزمان برقرار بود به این معناست که دو مستطیل فصل مشترک دارند:

1. ضلع بالایی مستطیل اول بالای ضلع پایینی مستطیل دوم باشد.
2. ضلع پایینی مستطیل اول پایین ضلع بالایی مستطیل دوم باشد.
3. ضلع چپی مستطیل اول چپ ضلع راستی مستطیل دوم باشد.
4. ضلع راستی مستطیل اول راست ضلع چپی مستطیل دوم باشد.

```
bool Rect::collides_with(Rect other) const
{
    return top() <= other.bottom() && bottom() >= other.top() &&
           left() <= other.right() && right() >= other.left();
}
```

کلاس Menu و استراکت MenuItem

این کلاس یک تجرید از کتابخانه ncurses می باشد که ساخت و نمایش منو را راحت تر می کند.

کانستراکتور آن عرض، عنوان منو، و یک لیست از گزینه های آن منو (**Menu Item**) دریافت می کند و با توجه به آنها یک نمونه از این کلاس می سازد.

فیلد *m_window* داده های مربوط به پنجره مجازی منو را نگه می دارد.

```
WINDOW *m_window;
```

این کلاس دو متد اصلی *start* و *stop* را داراست که به ترتیب نمایش منو را شروع و تمام می کنند.

```
/**
 * Start displaying the menu.
 *
 * Note that this method is a blocking call.
 */
void start();

/**
 * Stops displaying the menu by interrupting its infinite loop.
 */
void stop();
```

کلاس Game

وضعیت و همچنین عملکرد های اصلی بازی در این کلاس تعبیه شده که در اینجا برخی از آنها را بررسی می کنیم.

فیلد های *m_wstatus* و *m_wgame* به ترتیب داده های مربوط به پنجره های مجازی وضعیت بازی و صفحه بازی را نگهداری می کنند.

```
WINDOW *m_wstatus;
WINDOW *m_wgame;
```

متد *start* در ابتدا یک دشمن را به صورت تصادفی در صفحه بازی قرار می دهد و سپس لوپ اصلی بازی را شروع می کند. این لوپ شامل ترسیم بازی روی صفحه، به روز رسانی وضعیت بازی، ذخیره کردن بازی در حافظه و دریافت ورودی از کاربر می باشد.

```
void Game::start()
{
    if (m_enemies.size() == 0)
    {
        spawn_enemy_randomly();
    }

    while (true)
    {

        render();
        integrate();
        save();

        if (m_stopped || !input())
        {
            return;
        }
    }
}
```

متد *collide_bullets_with_enemies* برخورد گلوله ها را با دشمن ها بررسی می کند، که در صورت وجود برخورد گلوله ناپدید شده و یکی از جان های دشمن کم می شود. اگر که جان های دشمن تمام شود آن نیز ناپدید شده و بازیکن به اندازه آن امتیاز می گیرد.

```
void Game::collide_bullets_with_enemies()
{
    for (auto bullet_it = m_bullets.begin(); bullet_it != m_bullets.end();)
    {
        const Bullet &bullet = *bullet_it;

        bool should_erase_bullet = false;
        for (auto enemy_it = m_enemies.begin(); enemy_it != m_enemies.end();)
        {
            Enemy &enemy = *enemy_it;

            if (enemy.collides_with(Rect(Vec2(2, 1), bullet.translation())))
            {
                should_erase_bullet = true;
                enemy.take_damage(1);
                if (enemy.is_dead())
                {
                    m_player.take_score(enemy.points());
                    enemy_it = m_enemies.erase(enemy_it);
                    spawn_enemy_randomly();
                }

                break;
            }

            enemy_it++;
        }

        if (should_erase_bullet)
        {
            bullet_it = m_bullets.erase(bullet_it);
            continue;
        }

        bullet_it++;
    }
}
```

متد های *save* و *load* نیز وضعیت کنونی بازی را، به ترتیب، در فایل باینری state.dat ذخیره و از این فایل بارگیری می کنند.

```
/**
 * Saves the game state to a file.
 */
void save() const;

/**
 * Loads the game state from a file.
 */
void load();
```

کلاس Player

وضعیت و عملکرد های مربوط به بازیکن را در خود نگه می دارد که در اینجا برخی از آن ها را بررسی می کنیم.

فیلد *m_health* تعداد جان های باقی مانده سفینه خودی را نگه می دارد.

فیلد *m_score* تعداد امتیازات کسب شده بازیکن را نگه می دارد.

فیلد *m_translation* موقعیت کنونی بازیکن را نگه می دارد.

```
int m_health;
int m_score;
Vec2 m_translation;
```

متد های *save_to* و *load_from* وضعیت کنونی بازیکن را، به ترتیب، در استریم داده شده به صورت باینری ذخیره و از آن بارگیری می کنند.

```
/**
 * Saves the player state into `out` stream.
 */
void save_to(std::ofstream &out) const;

/**
 * Loads the player state from `in` stream.
 */
void load_from(std::ifstream &in);
```

کلاس Bullet

وضعیت و عملکرد های مربوط به تیر های شلیک شده را در خود نگه می دارد که در اینجا برخی از آن ها را بررسی می کنیم.

فیلد *m_translation* موقعیت کنونی تیر را نگه می دارد.

```
Vec2 m_translation;
```

متد های *save_to* و *load_from* نیز درست مانند همین متد ها در کلاس **Player** عمل می کنند.

کلاس Enemy

وضعیت و عملکرد های مربوط به دشمن ها را در خود نگه می دارد که در اینجا برخی از آن ها را بررسی می کنیم.

فیلد *m_health* تعداد جان های باقی مانده سفینه دشمن را نگه می دارد.

فیلد *m_collision* موقعیت و اندازه سفینه دشمن را نگه می دارد.

```
int m_health;
Rect m_collision;
```

متد های *save_to* و *load_from* نیز درست مانند همین متد ها در کلاس **Player** عمل می کنند.

متد points تعداد امتیاز های قابل کسب از نابود کردن این سفینه را بر می گرداند.

```
int Enemy::points() const
{
    return 2 * m_collision.size.y * m_collision.size.x;
}
```

ابزار و منابع

I. Git

سیستم کنترل نسخه (VCS) استفاده شده برای مدیریت پروژه.

II. GNU Make

ابزار مدیریت تولید فایل های خروجی استفاده شده جهت کاهش زمان و منابع مورد نیاز برای بیلد پروژه.

III. کتابخانه ncurses

یک کتابخانه برنامه نویسی برای ایجاد رابط کاربری مبتنی بر متن (TUI) که در این پروژه از آن بعنوان روش اصلی تعامل با کاربر استفاده شده است.

IV. C++ Reference

لینک: cppreference.com

استفاده شده برای مرور سیگنچر توابع کتابخانه های C++ و یادگیری توابع لاندا.

V. NCURSES Programming HOWTO

لینک: tldp.org/HOWTO/NCURSES-Programming-HOWTO

جهت یادگیری قابلیت های کتابخانه ncurses.

VI. Casual Coder از Ncurses Tutorials

لینک: youtube.com/playlist?list=PL2U2TQ__OrQ8jTf0_noNKtHMuYlyxQl4v

آشنایی با کتابخانه ncurses و استفاده کارآمد از آن.