# StarWars ver2.0

03.01.2024

Shayan Pakrad
Bu-Ali Sina university

BP Project

Final Project

# فهرست


**فهرست** **1**
**مقدمه** **1**
**هدف** **2**
**نحوه پیاده سازی** **2**
برنامه سازی به روش top-down design 2
رعایت کردن قوانین برنامه سازی ماژولار 2
برناه سازی به صورت data-driven 3
**توابع** **5**
تابع show_menu و start_game 5
تابع init_new_game 5
تابع play_game 6
تابع move_spaceship_down و move_spaceship_left و move_spaceship_right 6
تابع move_bullet 6
تابع check_spaceship_enemy_collision و check_bullet_enemy_collision 6
و check_enemy_border_collision 6
تابع create_enemy 7
تابع cell_to_string و render_map 7
تابع check_spaceship_health 7
تابع game_over 7
تابع check_points 7
تابع delete_saved_file و load_game و save_game 8
**تصاویر بازی** **8**
**گیت هاب** **9**
**استک اور فلو** **10**

# مقدمه

سلام!

شما در حال مطالعه گزارش کار پروژه پایانی درس مبانی برنامه سازی رشته مهندسی کامپیوتر شایان پاک راد هستید.

اسم این پروژه هم مثل پروژه قبلی starwars هست اما با کلی جزئیات متفاوت.

در واقع این پروژه یک بازی در محیط ترمینال یا کامندلاین می باشد که با استفاده از کاراکتر های کیبوردی پیاده سازی شده است.

در این گزارش سعی بر آن بوده که توضیح مختصر و مفیدی درمورد نحوه پیاده سازی این پروژّ داشته باشیم.

داکیومنت پروژه را می توانید در قسمت گیت هاب پروژه پیدا کنید.

# هدف

هدف از ارائه این پروژه برای دانشجویان در مبانی برنامه ساازی آشنایی با نحوه ساخت یک برنامه با بهره گیری از مفاهیم برنامه نویسی ساخت یافته و آشنایی با برنامه سازی فانکشنال و همچنین آشنایی با محیط ترمینال بوده است.

# نحوه پیاده سازی

## برنامه سازی به روش top-down design

در بخش پیاده سازی پروژه سعی کردیم از مفهوم top-down designاستفاده بکنیم. Top-down design به ما کمک می کند تا دید بسیار خوبی نسبت به روند پیاده سازی پروژه داشته باشیم و همچنین تمام فانکشن های مورد نیاز را از قبل بدانیم، به همین روش می توانیم یک خط مشی از کل رند پیاده سازی داشته باشیم.

## رعایت کردن قوانین برنامه سازی ماژولار

اگر به کد برنامه پروژه نگاهی بیندازیم متوجه استفاده خیلی زیاد از تابع های مختلف و تعریف تابع های متفاوت میشوید. در برنامه سازی ماژولار سعی بر آن است که تا آنجایی که می شود عملکرد های متفاوت (از لحاظ معنایی) یک برنامه را باه تابع های کوچکتر تقسیم کنیم بنابراین نگهداری و دیباگ کردن برنامه برای برنامه نویس بسیار راحت تر شده و توسعه برنامه آسان تر می شود.

```
void start_game();
int show_menu();
void init_new_game(Game &game);
void play_game(Game &game);
void move_bullets(vector<Bullet> &bullets);
void move_spaceship_left(Game &game);
void move_spaceship_right(Game &game);
void move_spaceship_down(Game &game);
void check_bullet_enemy_collision(Game &game);
bool check_spaceship_enemy_collision(Game &game);
void check_enemy_border_collision(Game &game);
Enemy create_enemy(int &map_size);
void render_map(Game &game);
string cell_to_string(int value);
void check_spaceship_health(int &spaceship_health);
void game_over();
void check_points(Game &game);
void save_game(Game &game);
void load_game(Game &game);
void delete_saved_file();
```

## برناه سازی به صورت data-driven

پیاده سازی این برنامه به صورت data-driven بوده و از مفاهیم شی گرایی در آن استفاده نشده است. این برنامه دارای struct های مختلف بوده و حتی از یک سری struct ها در stuct های دیگر استفاده شده.

```
struct Enemy{
    string name;
    int x;
    int y;
    int health;
    int size;
};
```

این struct مشخصات enemy را از جمله موقعیت، اسم، health و سایز نگه می دارد. در ادامه شاهد استفاده از این struct در struct های دیگر هستیم.

```
struct Spaceship{
    int x;
    int y;
    int health;
};
```

این struct دارای دیتای موقعیت و health سفینه خودی می باشد. در ادامه شاهد استفاده از این struct در struct های دیگر هستیم.

```
struct Bullet{
    int x;
    int y;
};
```

این نوع داده موقعیت گلوله های بازی راحفظ می کند

```
struct Game{
    int map_size;
    int point;
    int target_point;
    Spaceship spaceship;
    Enemy enemy;
    vector<Bullet> bullets;
};
```

این struct دارای تمام مشخصات بازی می باشد. وجود این struct به ما در save و load بازی بسیار کمک می کند و همچنین با وجود این struct دیتا های پاس داده شده میان توابع به صورت چشمگیری کاهش می یابد. یعنی به جای پاس دادن تمام اطلاعات بازی فقط این struct را پاس می دهیم

# توابع

در ادامه می خواهیم عملکرد توابع برنامه را توضیح دهیم.

## تابع start_game و show_menu

تابع show_menu مسئول نشان دادن منو و تابع start_game مسئول گرفتن ورودی و تصمیم در مورد ادامه روند بازی می باشد.

## تابع init_new_game

این تابع در صورت انتخاب گزینه start new game توسط کاربر فراخوانی می شود و مسئول مقدار دهی های اولیه است.

## تابع play_game

این تابع مسئول تعامل با کاربر می باشد که می تواند عمل های حرکت به چپ و راست، فایر اونلی و همچنین نشان دادن منو هنگام بازی را کنترل کند

## تابع move_spaceship_right و move_spaceship_left و move_spaceship_down

این توابع به ترتیب عملیات های حرکت به سمت راست و چپ و حرکت به سمت پایین را کنترل می کنند

حرکت به سمت پایین در بازی فقط با شلیک گلوله همراه است و تغییری در موقعیت سفینه ایجاد نمی کند.

داخل این توابع شاهد فراخوانی توابع زیادی هستیم که توابع پایین یکسری از آن ها می باشند

## تابع move_bullet

این تابع مسئول این است که با هر حرکت بازیکن تمام گلوله های موجود در نقشه را یک خانه به سمت بالا ببرد و یک گلوله جدید از روبروی سفینه شلیک کند و همچنینگلوله هایی که از مپ خارج می شوند را از بین ببرد

## تابع check_bullet_enemy_collision و check_spaceship_enemy_collision

## و check_enemy_border_collision

این توابع به ترتیب مسئول چک کردن برخورد گلوله ها با سفینه دشمن هستند و در صورت برخورد گلوله ها از health سفینه دشمن کم می کنند

تابع دوم وظیفه دارد برخورد سفینه دشمن با خودی را بررسی کند و در صورت برخورد سفینه دشمن را از بین ببرد و از health سفینه خودی کم کند

تابع سوم مسئول بررسی برخورد سفینه دشمن با پایین مپ می باشد یعنی در صورت پایین آمدن سفینه دشمن تا حدی که با سفینه خودی همسطر شود باعث کم شدن از health سفینه خودی و از بین رفتن سفینه دشمن می شود.

## تابع create_enemy

این تابع توانایی تولید و مقدار دهی های اولیه یک enemy جدید به صورت شانسی و برگرداندن آن با نوع داده ای enemy را دارد.

## تابع render_map و cell_to_string

تابع render_map مسئول چاپ کردن نقشه بازی در هر بار فراخوانی است و از تابع cell_to_string برای جایگذاری کاراکتر های مناسب در خانه های نقشه می باشد.

## تابع check_spaceship_health

این تابع وظیفه دارد در هر بار فراخوانی health سفینه خودی را چک کند و در صورت برابری با صفر تابع game_over را فراخوانی می کند

## تابع game_over

این تابع مسئول خاتمه دادن به رند بازی و خروج از محیط ترمینال می باشد

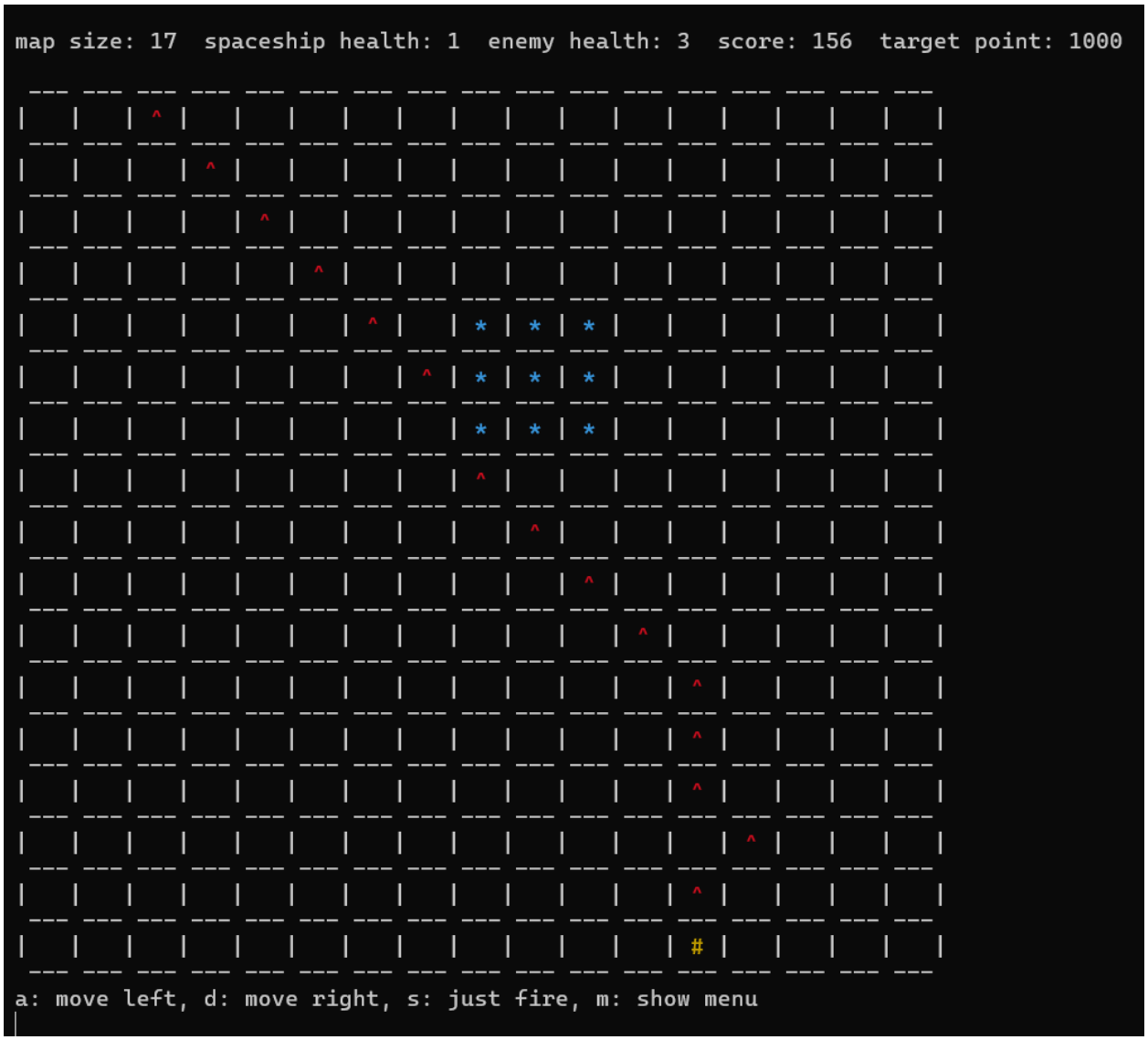
## تابع check_points

این تابع مسئول چک کردن امتیاز بازی و مقایسه آن با امتیاز مشخص شده می باشد که در صورت رسیدن به امتیاز از قبل مشخص شده وظیفه دارد به بازیکن اطلاع رسانی مربوطه را انجام دهد

### تابع save_game و load_game و delete_saved_file

این توابع به ترتیب مسئول ذخیرو و لود بازی و همچنین پاک کردن فایل سیو شده می باشند.

## تصاویر بازی




## گیت هاب

این پروژه دارای صفحه گیت هاب کامل بوده و کل روند پیاده سازی پروژه به صورت کامیت های متفاوت قابل مشاهده است

GitHub : https://github.com/Shayan-Pakrad/StarWars2

## استک اور فلو

Stack over flow : https://stackoverflow.com/users/22678991/shayan-pakrad