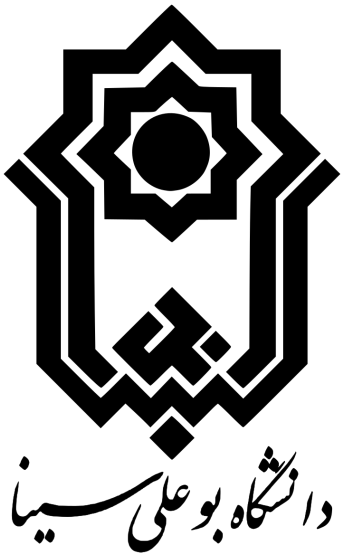
Star Wars



گزارش کار پروژه نهایی

02.22.2024

Pouya Omidi 40212358002

Pouya Tavakoli 40212358011

Basic programming winter 1402

Professor : Dr.H.Bashiri

فهرست

توابع

پروژه در یک نگاه

این پروژه یک بازی ساده است که شما در آن در نقش نجات دهنده دنیا دشمنان خود را در کهکشان نابود می کنید.

دشمنان شما چهار نوع هستند که هر یک اندازه و میزان سختی متفاوتی دارند.

در ابتدای بازی هر دشمن به صورت اتفاقی انتخاب می شود و با افزایش امتیاز شما و در نتیجه رفتن به مرحله های بالاتر ، بازی سخت تر می شود و دشمنان ساده دیگر نمایش داده نمی شوند.

داده های اصلی بازی در یک فایل ذخیره می شوند و در دفعه بعدی که بازی اجرا می شود قابل دسترسی هستند.

اهداف پروژه

در این پروژه سعی شده در حین پیاده سازی یک بازی ساده اصول زیر رعایت شود :

ماژولاریتی :

بازی از تعدادی تابع ساخته شده که هر یک عملکرد یکتا و مستقلی دارند. اکثر توابع فقط یک کار را انجام می دهند.

کار با فایل ها :

شما با خروج از بازی ، بازی قبلی را از دست نمی دهید و می توانید هر زمان که خواستید برگردید و ادامه بازی را پیش ببرید.

کار گروهی :

پیاده سازی این پروژه به صورت گروهی و با استفاده از گیت و بر بستر گیت هاب بوده است.

با کلیک بر روی لوگو زیر می توانید روند ساخت بازی را مشاهده کنید .

همانطور که گفته شد بازی از چند تابع تشکیل شده و برای دسته بندی اطلاعات از استراکت ها استفاده شده که در ادامه به بررسی هر یک می پردازیم.

استراکت

با توجه به اینکه برنامه نیاز داشت تا اطلاعات کلیدی از تابعی به تابع دیگر انتقال پیدا کند و حجم این اطلاعات زیاد بود تصمیم گرفتیم از استراکت استفاده کنیم تا تا کد ما ساختاری منظم داشته باشد .

از طرفی اطلاعات موجود در استراکت به هم وابسته هستند و روند بازی با هم تاثیر گذارند پس استراکت گزینه مناسبی بود .

GameData

این استراکت اطلاعات بازی را نگهداری میکند که شامل مرحله ، امتیاز ، امتیاز نهایی ، وضعیت بازی و تعداد دشمن های کشته شده است.

Dart , Striker , Wraith , Banshee

این چهار استراکت نگهداری اطلاعات 4 نوع دشمن را بر عهده دارند که این این اطلاعات شامل اندازه ، تعداد تیر مورد نیاز برای کشته شدن تعداد موجود از این دشمن در نقشه و موقعیت مکانی هستند.

ممکن است سوال پیش بیاید که چرا همگی را تبدیل به یک استراکت نکردیم و آن را enemy بنامیم !؟ حقیقت ماجرا این هست که ما به این فکر کردیم که هر دشمن متفاوت از دشمن دیگری است و ویژگی های متفاوتی دارد پس نمی توان ان ها را یک نوع شئ دید، بنابراین برای هرکدام استراکت جدا گانه ای درست کردیم .

spaceShip

این استراکت موقعیت و heal سفینه خودی را نگهداری می کند .

با توجه به اینکه سفینه خودی فقط در پایین ترین ردیف حرکت می کند این استراکت موقعیت افقی را ذخیره می کند.

Bullet

این استراکت صرفا موقعیت گلوله را ذخیره میکند .

تابع ها

قبل از آن که توضیحات را شروع کنیم لازم به ذکر است که در این گزارش ما توابع را بر اساس concept طبقه بندی و توضیح می دهیم که بتوانیم درک خوبی از نحوه عملکرد بازی داشته باشیم .

تابع اصلی

```
int main()
```

در تابع اصلی ما فقط و فقط تابع menu صدا زده میشود تا بازی شروع شود و سایر عملکرد های برنامه به عهده توابع دیگر است .

تابع های ساختاری بازی

```
int menu()
```

تابع منو یک تابع چند منظوره برای نمایش دادن انواع مختلف منو در قسمت های مختلف بازی است. ورودی تابع مشخص کننده این است که منو چه زمانی فراخوانی شده است.

اگر منو در شروع بازی فراخوانی شود گزینه های زیر را خواهد داشت :

- شروع بازی جدید
- ادامه بازی قبلی
- خروج

همانند بسیاری از بازی ها در این بازی باز کردن منو در واقع pause کننده بازی است و منو در این حالت گزینه های زیر را خواهد داشت :

- ادامه بازی
- خروج

`int startGame()`

این تابع بازی را با شرایط داده شده شروع می کند و به دو صورت قابل فراخوانی است . تابع می تواند بازی جدید را شروع کند یا ادامه بازی قبلی را اجرا کند . در هر دو مورد عملکرد تابع به صورت فراخوانی چند تابع با داده های مورد نیاز آن ها است تا بازی شروع شود.

شروع بازی جدید :

تابع اندازه مورد علاقه کاربر برای نقشه را می گیرد و تابع `print map` را فراخوانی می کند

ادامه بازی قبلی :

تابع با فراخوانی تابع `gameloader` داده های فایل را استخراج میکند و با توجه به داده های دریافت شده از فایل وضعیت های زیر را بررسی می کند .

بازی قبلی به اتمام رسیده : با توجه به اینکه نتیجه بازی قبلی برنده یا بازنده بوده پیغام مربوط به آن به کاربر نمایش داده می شود. و کاربر به منو بازی هدایت می شود.

بازی قبلی هنوز به اتمام نرسیده : ادامه بازی قبلی با اطلاعات دریافت شده از فایل اجرا می شود.

بازی قبلی وجود ندارد : پیامی در رابطه با اینکه بازی سیو نداریم نمایش داده می شود و سپس به منوی اصلی برای شروع بازی جدید هدایت می شود.

نکته قابل توجه : در این قسمت همانطور که قابل مشاهده است ، ما وکتور هایی از انواع دشمن ها داریم . دلیل این موضوع این است که وقتی شرایطی بوجود می آید که دشمن ما باید از بین برود بتوانیم با دستور های ساده ی وکتور آن را از حافظه پاک کنیم و از طرفی هم برای بخش امتیازی پنجم ابتکار خوبی بود که به دلیل کمبود وقت انجام نشد .

تابع های سازنده مختصات اشیاء متحرک بازی

`void Enemy_coordinate_maker()`

این تابع در آغاز کار یک عدد تصادفی ایجاد می کند که نوع دشمن را تعیین میکند سپس چک می کند که آیا دشمن در نقشه وجود دارد یا خیر. سپس با توجه به عدد تصادفی مختصاتی برای دشمن تعیین می کند.

همچنین در این قسمت بخش امتیازی هفتم هم لحاظ شده است ، در هر قسمتی که شرط ها وجود دارند ،شرط آن است که اگر امتیازی از حد تعریف شده بیشتر بود اون نوع دشمن ساخته نشود و این عمل آنقدر تکرار شود تا دشمن از انواع دیگر ساخته شود.

توجه : این حد ها با توجه به سند پروژه به دلخواه طراح های بازی تعیین شده است.

```
void new_bullet_maker()
```

این تابع در موقعیت سفینه خودی یک تیر اضافه می کند .به صورتی که به نظر می رسد تیر از سفینه خارج شده.

در این قسمت مختصات گلوله جدید ساخته شده و در وکتوری از استراکت bullet ذخیره میشود تا در بازی از آن استفاده شود .

تابع نمایش دهنده مپ بازی

```
void printMap()
```

نحوه کار این تابع به کمک یک آرایه دوبعدی و چند flag است . تابع ابتدا مختصات سفینه و دشمن ها و تیر ها را می گیرد و flag ها را مشخص می کند . به عنوان مثال وکتور Dart را بررسی میکند و اگر این نوع دشمن وجود داشت flag آن برابر true قرار داده میشود و این عمل برای انواع دشمن ها صورت میگیرد و اگر در مختصات مورد نظر دشمنی وجود نداشت آن را خالی می گذارد و همین استدلال برای گلوله ها نیز برقرار است و خونه به خونه چاپ می کند تا نقشه بازی در انتها بصورت کامل نمایان میشود . و برای هر عضو آرایه با توجه به flag مربوط به آن ، کاراکتر و رنگ مربوط به آن را چاپ می کند.

تابع های حرکت های بازی

```
void move_right() , void move_left()
```

همانطور که از اسم این دو تابع مشخص است ، کار این توابع حرکت دادن سفینه خودی است . این تابع ها زمانی که فراخوانی شوند مختصات سفینه خودی را به صورتی تغییر می دهند که سفینه یک خانه به راست و چپ برود.

برای حفظ ماژولاریتی ، عملیات دریافت ورودی که مشخص کننده نوع حرکت است در این تابع ها انجام نمی شود بلکه این تابع ها در زمان مورد نیاز فراخوانی می شوند و مختصات سفینه را تغییر می دهند.

```
void move_bullets()
```

این تابع مختصات تیر های موجود در صفحه را به صورتی تغییر می دهد که یک خانه به سمت بالا حرکت کنند . همانند تابع های move left , right این تابع نیز فقط تغییر دهنده مختصات است .

همچنین در این قسمت با فراخوانی تابع `Bullet_outofBound` که در ادامه توضیح داده خواهد شد ، بررسی میکنیم که اگر گلوله ما از مپ بازی خارج شد آن را حذف کند در غیر اینصورت یک واحد آن را به سمت بالا ببرد.

`void move_enemies_down()`

این تابع مختصات دشمن موجود در صفحه را به صورتی تغییر میدهد که یک خانه به سمت پایین حرکت کند و همانند سایر تابع هایی که در این قسمت توضیح داده شد این تابع هم فقط تغییر دهنده مختصات است .

در این قسمت صرفا مختصات های دشمن موجود را یک واحد افزایش می دهد که به این منظور دشمن یک واحد به سمت پایین حرکت میکند.

تابع های کنترل کننده مختصات و ویژگی های اشیاء بازی

`bool bullet_outOfBound()`

در این تابع صرفا وکتور bullet به تابع ارسال شده و با چک کردن آن که گلوله مورد نظر خارج از مپ بازی هست یا نه ، true و false بودن آن را باز می گرداند تا در تابعی که در قسمت قبل گفته شد استفاده شود.

`bool enemy_outOfBound()`

در این تابع وکتور های تمامی دشمن ها و همچنین n که نشان دهنده تعداد سطر و ستون است ارسال میشود به تابع . حال این تابع چک میکند که اگر مختصات آخرین سطر از دشمن مورد نظر خارج از مپ بازی بود مقدار true را باز میگرداند و اگر نبود مقدار false را باز میگرداند.

کاربرد این تابع را می توان در تابع `removeAndRespawnEnemy` مشاهده کرد .

`bool collision()`

این تابه به این منظور طراحی شده که چک کند که آیا دشمن با سفینه خودی برخورد میکند یا نه . به این منظور وکتور های دشمن ها ، وکتور سفینه خودی و n به تابع ارسال می شود.

اگر برخورد داشته باشد True را باز می گرداند و در غیر اینصورت False را .

کاربرد این تابع را نیز می توان در تابع `remove And Re spawn Enemy` مشاهده کرد .

`Void removeAndRespawnEnemy()`

در این تابع با بررسی شرط های لازم دشمن مورد نظر را حذف و دشمن جدیدی را وارد مپ بازی میکنیم.

به اینصورت که برای هر نوع دشمن تابع هایی که در قسمت های قبلی آن ها را توضیح دادیم صدا زده میشوند و در صورتی که هر کدام از آنها مقدار True را بازگردانند و همچنین جون دشمن ما تمام نشده باشد ، آن دشمن را از وکتور حذف ، از جون سفینه خودی کم و در نهایت با فراخوانی تابع ساخت دشمن یعنی تابع

`enemy coordinate maker` دشمن جدیدی را میسازد.

```
void enemy_heal_check() , bool isHit ()
```

این تابع به ردیف آخر مپ برخورد کند یا به سفینه خودی برخورد کند دشمن را حذف و یک دشمن جدید جایگزین کند.

دو کار را همزمان انجام میدهد. اول آنکه با فراخوانی توابع isHit بررسی میکند که اگر گلوله های موجود در صفحه ما برخوردی با دشمن موجود در صفحه داشته اند آن گلوله هایی را که برخورد کرده اند را از بین میبرد و همچنین از جون دشمن موجود در صفحه کم میکند.

نکته مهم : ابتکاری که در این قسمت بکار رفته به این صورت است که چون قصد حذف گلوله ها را از وکتور داریم باید از دستور erase استفاده کنیم اما این دستور موقعیت را میگیرد و آن را از داخل وکتور پاک میکند پس باید جوری عمل کنیم که موقعیت گلوله برخورد کرده را به این دستور بدهیم . بنابراین به جای آنکه مقدار های موجود در وکتور bullet را بصورت مستقیم مورد بررسی قرار دهیم باید آدرس آنها را که بصورت پوینتر هست بررسی کنیم ، پس تابع جداگانه ای تحت عنوان isHit برای هر دشمن درست کرد کردیم که مقدار درون پوینتر که همان مختصات گلوله ما هست را مورد بررسی قرار دهد و اگر برخوردی داشتیم به ما مقدار True را بازگرداند که در این صورت شرط انجام کارهایی که گفته شد برقرار است پس با یه تیر دو نشون زدیم ، هم موقعیت را داریم و میتوانیم به راحتی گلوله را از وکتور حذف کنیم و هم جون دشمن کم میشود.

```
void enemy_damage_check()
```

این تابع صرفا وظیفه حذف دشمن را که جون آن به اتمام رسیده است را دارد که برای همین به این تابع وکتور های دشمن ، n و همچنین داده gamedata داده شده است.

تابع به اینصورت کار میکند که اگر جون دشمن موجود در صفحه تمام شده باشد آن را از وکتور پاک میکند و همچنین با توجه به نوع دشمن امتیاز آن را به امتیاز کل و امتیازی که برای افزایش level (قسمت امتیازی اول) دربازی نیاز داریم اضافه میکند و در نهایت دشمن جدیدی را توسط تابع enemy_coordinate_maker میسازد.

تابع نشان دهنده وضعیت سفینه خودی

```
bool shipstatus()
```

این تابع صرفا بررسی میکند که آیا جون سفینه خودی تمام شده است یا خیر و با توجه به آن True یا False را برمیگرداند. کاربرد آن را می توان در تابع startgame مشاهده کرد.

در واقع یکی از شرط های لازم برای ادامه بازی است و اگر مقدار آن False برگردانده شود بازی تمام می شود .

تابع مربوط به leveling

```
Void update_level()
```

این تابع با هر ۲۰۰ امتیازی که ما بدست می آوریم یک لول به لول ما می افزاید. نحوه ی عملکرد آن به سادگی گفتنش هست:)

تابع های refresh

```
void refresh()
```

تابع رفرش مجموعه ای از تابع های بازی است که بازی را به state بعدی می برند .وجود این تابع خوانایی کد را به صورت قابل توجهی بالا می برد و از تکرار یک مجموعه طولانی تابع جلوگیری می کند اگر بازی را همانند چندین اسلاید پشت سر هم تصور کنیم برای رفتن به اسلاید بعدی نیاز داریم که چندین تابع را فراخوانی کنیم تا داده های بازی را با توجه به وضعیت فعلی بررسی و برای وضعیت بعدی آماده کنند.

بازی زمانی نیاز به رفرش شده دارد که حرکتی انجام شود پس تابع رفرش زمانی فراخوانده خواهد شد که کاربر دستور حرکت را به برنامه بدهد.

تابع رفرش شامل عملکرد های زیر است که هر یک به صورت مجزا توسط یک تابع انجام می شوند :

- چک کردن heal دشمنان
- حذف کردن دشمن در صورتی که heal صفر شود
- ساخت یک تیر جدید
- یک خانه پایین آوردن تیر ها
- بررسی مرحله بازی
- سیو کردن بازی در فایل

```
bool ref2()
```

این تابع مسئولیت کنترل کردن یک شرط خاص و مهم در تابع رفرش اصلی را بر عهده دارد.

بزارید با یک مثال توضیح دهم تا متوجه کارکرد این تابع شوید . اگر تابع refresh صدا زده شود و جون دشمن موجود در صفحه تمام شده باشد دشمن حذف و دشمن جدیدی ساخته میشود ولی هنوز در چون تابع printmap صدا زده نشده این دشمن را بر روی صفحه نمی بینیم ،از طرفی در خط های بعدی از تابع رفرش می بینیم که تابعی که یک واحد دشمن را به پایین انتقال میدهد صدا زده شده است ، اتفاقی که می افتد این است که به دلیل اتفاقات بالا زمانی که تابع printmap صدا زده می شود ، دشمن از ردیف دوم شروع به حرکت میکند ،حال برای اینکه این اتفاق نیفتد تابع رفرش دوم را ساختیم.

عملکرد ان به این صورت است که زمانی که جون دشمن چک شد این تابع صدا زده میشود و در داخل این تابع چک میشود که اگر دشمن جونش تمام شده به ما مقدار True را بازگرداند که به این معنی است که قراره دشمن جدیدی ساخته شود پس باید دستوری که دشمن را یک واحد به سمت پایین می برد اجرا نشود که دچار باگ در بازی نشویم. بنابراین شرط را این قرار میدهیم که فقط اگر دشمن جونش تمام نشده بود دستور یک واحد پایین آمدن دشمن ها اجرا شود.

تابع های مربوط به فایل

یکی از مهم ترین بخش های این پروژه کار با فایل است که در اینجا به کمک دو تابع انجام می شود.

- Game saver
- Game loader

```
void gamesaver()
```

این تابع یک فایل به اسم "gameData.txt" ایجاد می کند و اطلاعات بازی را در آن ذخیره میکند .

اطلاعاتی در فایل ذخیره می شوند که ما می توانیم هر زمان در آینده با استفاده از این داده ها و فقط همین داده ها بازی را دقیقا در همان حالتی که قبلا بوده بازسازی کنیم. این داده ها عبارتند از :

- نتیجه بازی win / loose / undefined
- اندازه مپ بازی که مقدار دلخواه کاربر در شروع بازی بوده است
- مرحله
- امتیاز
- موقعیت افقی سفینه خودی

- میزان heal سفینه خودی
- تعداد دشمن کشته شده
- امتیاز هدف
- کاراکتر نمایشی سفینه خودی
- نام و مختصات و جون باقی مانده دشمن

```
int gameloader()
```

این تابع فایل " gameData.txt" را باز میکند و شروع به خواندن اطلاعات میکند و کار های زیر را با توجه به اطلاعات موجود در فایل انجام می دهد.

ابتدا نتیجه بازی را میخواند و آن را به صورت یک عدد که برای تابع game loader قابل فهم است اعلام می کند.

برای بهینگی بیشتر اگر بازی قبلی تمام شده بود تابع game loader همانجا return می شود و ادامه فایل را بررسی نمی کند ولی اگر بازی قبلی قابل ادامه دادن بود تابع ادامه پیدا می کند و اطلاعات بازی قبلی را می خواند و در متغیر های بازی جایگذاری می کند .

خواندن اطلاعات تا اسم دشمن به صورت عادی ادامه پیدا میکند پس از خوانده شده اسم دشمن ، تابع نوع دشمن را تشخیص می دهد و با توجه به نوع دشمن داده های موجود در فایل را در متغیرهای مربوط به آن جایگذاری می کند.

تابع مربوط به نمایش اطلاعات دشمنان از بین رفته

```
void summary()
```

این تابع صرفا یک کار ساده را انجام می دهد آن هم نشان دادن نوع دشمن و در جلوی آن تعداد از بین رفته ی آن و همچنین تعداد کل دشمن های از بین رفته است.

ERROR HANDLING

استفاده از error handling در چهار قسمت از برنامه امکان اطمینان از پایداری و عملکرد صحیح برنامه فراهم می کند. به عنوان مثال وقتی کاربر ورودی نا معتبری وارد میکند ، برنامه به کاربر اعلام می کند که ورودی invalid است و یا اتفاقی نمی افتد (در ادامه دلیل این موضوع توضیح داده خواهد شد) . اما بریم سراغ هر کدام از این قسمت ها :

اولین جا تابع menu هست :

در این تابع ما یک switch case داریم که دارای دو حالت است که هر کدام نیز حالات خود را دارند . در switch case ما هیچوقت ورودی اشتباهی دریافت نمی کنیم که بخواهیم اروری برای ما نمایش دهد زیرا کاربر هیچ ورودی که مستقیما با خود switch case در ارتباط باشد ندارد و ما در داخل برنامه خودمان از آن استفاده کردیم و با فراخوانی های متعدد و بررسی ساختار برنامه مطمئن شدیم که به درستی کار میکند .

اما داخل هر case ما ورودی هایی را از کاربر میگیریم پس اینجا مشخصا امکان وارد کردن ورودی اشتباه از کاربر وجود دارد ، در قسمت اول ما سه نوع ورودی داریم که با وارد کردن هرکدام عملکرد مناسب اتفاق می افتد اما اگر چیز دیگری وارد شود برنامه در منوی موجود در صفحه میماند تا کاربر ورودی درست را وارد کند ، دلیل اینکه پیامی در ارتباط با وارد کردن ورودی اشتباه نمایش داده نمی شود به این دلیل است که اگر تا بحال بازی انجام داده باشید متوجه می شوید که شما داخل هر منویی که باشید و هر چیزی جز ورودی های مد نظر را وارد کنید خطایی به شما نمی دهد و صرفا زمانی که گزینه های موجود را انتخاب کنید عملکرد های مناسب را انجام میدهد.

در کیس دوم هم به همین تربیت است ، اگر شما چیزی جز ورودی ها وارد کنید خطایی در آن جهت به شما نشان نمیدهد اما منتظر میماند تا شما ورودی صحیح را وارد کنید و سپس عملکرد مناسب آن را انجام میدهد.

دومین جا تابع startgame هست :

در این قسمت ما ورودی x را داریم که نتیجه انتخاب ما در منوی اصلی است و با توجه به اینکه در قسمت قبل ما فرایند error handling را انجام دادیم اینجا نیازی به انجام این فرایند نیست و ورودی اشتباهی وارد نمیشود، اما در داخل آن زمانی که x برابر یک هست که به معنای شروع بازی جدید است شرط آنکه سایز مپ بازی کوچک تر از ۱۶ نباشد بررسی شده و اگر عددی کمتر از وارد شود به کاربر پیامی مربوط به ان را نمایش میدهد و از او میخواهد که عدد بیشتر از ۱۶ را وارد کند .

زمانی که x برابر ۲ هست ، شرط هایی بررسی میشود اما نیازی به error handling نیست زیرا این فرایند در تابع gameloader صورت میگیرد و در اینجا ما ورودی اشتباهی نخواهیم داشت که پیامی مربوط به ان را نمایش دهیم.

در همین تابع ما برای حرکت دادن و تیر زدن چهار حرف را تعریف کرده ایم و اگر چیزی جز این چهار حرف را کاربر وارد کند خطای مربوط به آن نمایش داده میشود .

همچنین در قسمت بعد از آن زمانی که کاربر حد نصاب امتیاز مشخص کرده در اول بازی را میگذراند از او سوالی میشود که جواب او از دو حالت خارج نیست و اگر کاربر چیزی جز اون دو حالت وارد کند ، خطای مربوط به آن نمایش داده میشود .

قسمت سوم تابع gamesaver هست :

در این قسمت تنها ایرادی که ممکن است پیش بیاید باز نشدن فایلی است که در آن می خواهیم سیو کنیم که در اینصورت ارور مناسبی برای آن نمایش داده میشود .

قسمت چهارم تابع gameloader هست :

اولین اروری که باید بررسی میشد همان ارور قسمت قبلی که باز نشدن فایل هست بود که همانند قبلی ارور مناسب نمایش داده میشود.

اما این قسمت چون در رابطه با خواندن بازی هست چالش های متعددی بوجود آورد که تمام آنها برطرف شد. قسمت اول این بود که تابع اولین کاری که انجام میدهد خواندن وضعیت بازی است یعنی win یا loose یا undefined که تابع با خواندن هر کدام از ان ها وضعیت را مشخص میکرد و تابع startgame میفرستاد اما سوال این بود که ایا ممکنه از این چهار حالت خارج باشه ؟ جواب خیر است زیرا برنامه در اولین حرکتی که شما انجام میدهید سیو میشود که در این صورت وضعیت undefined ثبت میشود و بعد از هر حرکت ممکن است این وضعیت تغییر کند ولی هیچگاه از این سه حالت خارج نیست. مورد بعدی عدم وجود نوع دشمن است که باز هم امکان ندارد زیرا هرگاه دشمنی از بین برود حتما و قطعا دشمن جدیدی بعد از حذف آن درست میشود پس با هر بار سیو کردن ما نوع و مختصات دشمن را در فایل ذخیره داریم .

بخش آخرش هم این بود که ما تعداد گلوله های درون مپ را نمیدانیم پس بنابراین نوشتن آنها را در آخر گذاشتیم تا در اینصورت وقتی میخواهیم مختصات آنها را بخوانیم بگوییم تا جایی که میتونی مختصات گلوله بخونی ادامه بده پس در اینصورت مشکلی در خواندن گلوله ها هم بوجود نمی آید.

نکته مهم : ممکن است سوال شود که چرا برای خارج شدن سفینه خودی از مپ بازی ارور مناسب نمایش داده نمیشود ؟!

جواب خیلی واضح است زیرا اصلا خارج نمیشود !!! بازی ما محدوده حرکاتش مشخص است و در واقع یک بازی open world نیست که به ما اجازه خارج شدن از محدوده ای را تحت شرایطی بدهد یا ندهد . همچنین در بازی هایی که اینگونه هستند شما هیچ اروری تحت عنوان اینکه از بازی خارج می شود دریافت نمی کنید زیرا همانطور که گفتم تمامی حدود مشخص هستند .

Git

کنترل ورژن و کار گروهی روی این پروژه به کمک گیت و گیت هاب انجام شده است.

مزایا:

در دست داشتن سابقه فایل ها

کار گروهی

مشکلات :

برخی مواقع تغییرات همخوانی نداشت که منجر به یادگیری بیشتر دستورات ویژه تر در گیت شد.

```
D:\starwars-v2>git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), 608 bytes | 1024 bytes/s, done.
From https://github.com/pouyatavakoli/starwars-v2
   5b8b478..711030c  master     -> origin/master
error: Your local changes to the following files would be overwritten by merge:
        main.cpp
Please commit your changes or stash them before you merge.
Aborting
Updating 5b8b478..711030c

D:\starwars-v2>git reset --hard origin/master
HEAD is now at 711030c Error Handling in each function is done , The infinite loop bug in startgame function is fixed
```