پروژه نهایی

بازی پک–من

درس برنامه سازی پیشرفته

استاد درس

دکتر مرتضی یوسف صنعتی

دانشجو

عادل عسگری

۴۰۰۱۲۳۵۸۰۲۷

بهار ۱۴۰۱


بسم الله الرحمن الرحیم

فهرست مطالب

۱ مقدمه

در این پروژه بر روی ساخت بازی قدیمی پک-من [۱] کار کرده ایم. بازی پک-من یک بازی ارکید محبوب و نوستالژی می باشد. این بازی محبوب جهانی ساخته ی استودیوی BANDAI NAMCO می باشد. و در ابتدا برای کنسول های قدیمی میکرو و سگا طراحی و ارایه شده بود. ولی به دلیل محبوبیت بالا حالا برای دستگاه های اندرویدی هم ارایه شده است. در این بازی بازیکن در مسیر های معینی حرکت می کند و باید مواظب دشمنان باشد. بازیکن در طول مسیر نیز به جمع اوری سکه و امتیاز می پردازد. زبان برنامه نویسی مورد استفاده در این پروژه c++ به همراه کتاب خانه SFML[۲] می باشد و در ابزار VSCODE نوشته شده است .

---

[۱] Pac-man

[۲] Simple and Fast Multimedia Library

۲ تعریف کلی بازی

۲ – ۱ مراحل

با توجه به کمبود وقت ، برای این بازی تنها یک مرحله در نظر گرفته شده است اما تمام قواعد بازی در این مرحله پیاده سازی شده است.

۲ – ۲ حرکت

برای حرکت دادن پک-من از کلیدهای جهت دار یا کلیدهای (W-A-S-D)[۳]استفاده می کنیم. با زدن کلید های حرکتی پک من به حرکت در آمده و به اطراف می رود. اما باید در نظر گرفت که حرکت زمانی امکان پذیر است که به دیوار ها برخورد نکرده باشیم. دیوار ها مسیر بازی را ترسیم می کنند و از عبور روح ها و پک من از روی دیوار ها جلوگیری می کنیم.

۲ – ۳ خوراک عادی

در محیط بازی یک سری نقطه وجود دارد که با خوردن[۴]هر کدام ۱۰ امتیاز به مجموع امتیازات پک-من افزوده می شود. در صورتی که تمام این نقاط توسط پک-من خورده شود بازی با برد به پایان می رسد و می توان به مرحله بعدی رفت.

۲ – ۴ خوراک قدرتی

در محیط بازی نقاط دایره ای توپر سفید رنگی قرار گرفته است که با خوردن آن ها ۵۰ امتیاز به مجموع امتیازات افزوده می شود. بلافاصله پس از خوردن خوراک قدرتی روح ها به مدت ۶ ثانیه[۵]در وضعیت ترسیده قرار گرفته و آبی رنگ می شوند و با چرخش ۱۸۰ در جه ای سعی می کنند تا از مسیر قبلی دور شوند. قبل از اتمام ۶ ثانیه ۵ بار[۶]رنگ روح های ترسیده بین سفید و آبی تعویض می شود.(حالت چشمک زدن بین رنگ آبی و سفید)

---

[۳] کلید W برای حرکت به بالا ، کلید A برای حرکت به چپ ، کلید S برای حرکت به پایین و کلید D برای حرکت به سمت راست مورد استفاده قرار می گیرد.

[۴] در واقع در بازی خوردن اتفاق نمی افتد و با رسیدن پک- من به مختصات هر نقطه فرض می کنیم پک-من آن نقطه را خورده و دیگر آن نقطه نمایش داده نمی شود. و امتیاز آن نقطه را برای بازیکن منظور می کنیم.

[۵] زمان در نظر گرفته شده ، مطابق جدول ارائه شده در زمان بندی پروژه برای مرحله اول می باشد.

[۶] تعداد در نظر گرفته شده مطابق جدول ارائه شده در جدول زمان بندی پروژه می باشد.

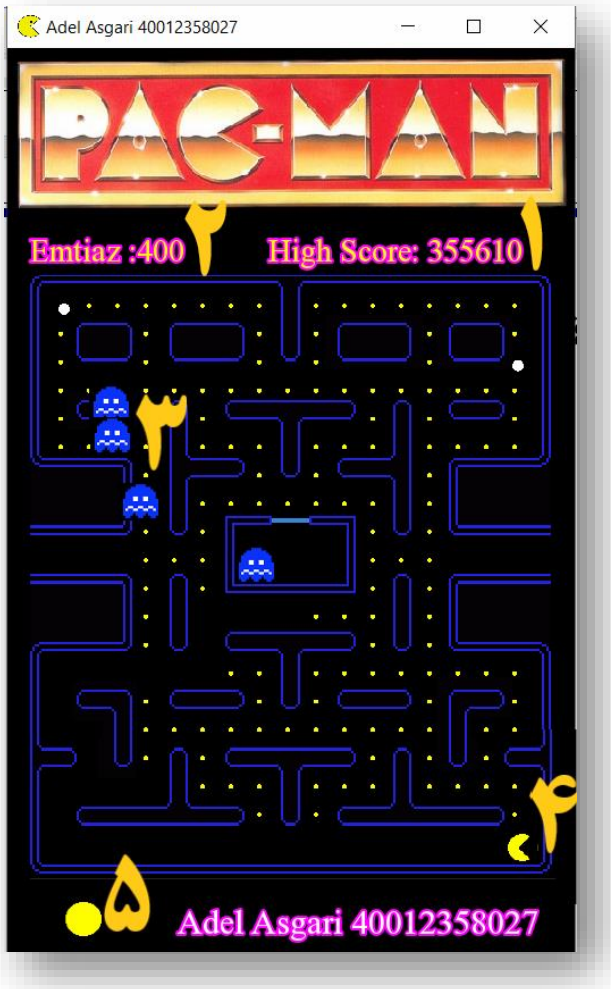
۲ – ۵ میوه

در مراحل مختلف بازی پس از خوردن تعدادی خوراک عادی میوه هایی به عنوان جایزه نمایش داده می شوند که ۱۰ ثانیه بر روی تصویرو در مختصات تصادفی ظاهر می شوند و با خوردن هر کدام امتیاز مشخصی به پک-من تعلق می گیرد

شکل ۲-۱ میوه گیلاس ظاهر شده در صفحه پس از خوردن ۷۰ خوراک عادی

۳ معرفی بازی

این بازی مطابق با بازی قواعد تعریف شده در جزوه پروژه پیاده سازی شده است. در تصویر زیر محیط بازی را مشاهده می کنیم.



شکل ۳-۱ محیط بازی پیاده سازی شده

بخش های شماره گذاری شده در بازی به شرح زیر می باشد:

۱-بیشترین امتیاز کسب شده تا این لحظه در بازی می باشد. به محض اینکه این بازی به پایان برسد(برد یا باخت) بیشترین امتیاز به روز رسانی می شود.[۷]

[۷] توضیحات تکمیلی در بخش مجزایی در قسمت کدها خواهد آمد.

۲-امتیاز این بازیکن در این بازی تا این لحظه این امتیازا را کسب نموده است. امتیاز از طریق خوردن خوراکی های عادی ، خوراکی های قدرتی، خوردن روح ها و تغذیه از غذاهایی که به عنوان پاداش به ما تعلق می گیرد، کسب می شود.

۳-روح های موجود در بازی ۴ عدد می باشند که در این عکس در حال ترسیده قرار دارند.

۴-پک-من یا بازیکن اصلی می باشد که با استفاده از کلیدهای جهت دار یا (W-A-S-D) در صفحه حرکت می کند.

۵- تعداد جان های باقی مانده پک من در این بخش قابل مشاهده است. تعداد جان های پک-من ۳ عدد می باشد که در ابتدای بازی یکی از جان ها در حال بازی می باشد و دو عدد دایره زرد رنگ در پایین صفحه نمایانگر تعداد جان های باقی مانده می باشد. با هر بار برخورد پک-من به روح هایی که در حالت تعقیب یا پرسه زدن باشد یکی از جان ها کم شده و ۲۰ امتیاز نیز کسر می شود.

۴ گرافیک پروژه

۴ – ۱ پنجره بازی

برای این بازی یک پنجره ۸۰۰ * ۵۰۰ در نظر گرفته شده است. تمام ترسیمات بازی بر روی این پنجره انجام می شود. محدودیت حداکثر ۶۰ فریم در ثانیه نیز برای این پنجره اعمال شده است.

```
RenderWindow MainWindow(VideoMode(500, 800), "Adel Asgari 40012358027");
```

۴ – ۲انیمشین بازی

برای ساخت انیمیشن های بازی از اشیایی از جنس spirit استفاده شده است. کافی است در فریم های مختلف برش هایی از یک عکس ثابت که حالت های مختلف یک کاراکتر را دارا می باشد را نمایش دهیم در نتیجه باعث می شود تا حرکت های یک کاراکتر مشابه یک انیمیشن دیده شود[۸]

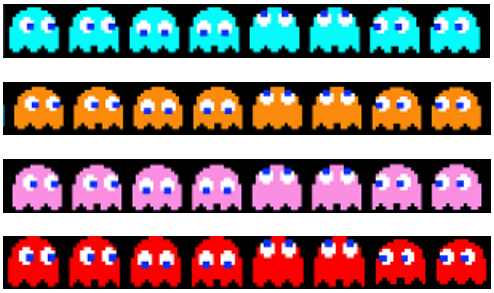
انیمیشن های کاراکترهای مختلف در وضعیت های متفاوتی اجرا می شوند. انیمیشن پک-من دائم تکرار می شود و با چرخش پک-من کاراکتر نیز می چرخد.

شکل ۴-۱ اسپرایت مورد استفاده برای انیمیشن پک-من

شکل ۴-۲ اسپرایت مورد استفاده برای باخت پک-من

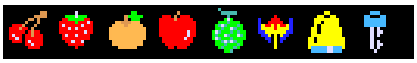
انیمیشن مورد استفاده در روح ها نیز با کلیدهای جهت دار اجرا می شوند. زمانی که روح به هر سمتی می چرخد کاراکتر روح به شکلی تنظیم می شود که چشم های روح به آن سمت حرکت نماید. همچنین برای هر سمتی دو عکس در نظرگرفته شده است که حالت پا ها متفاوت می باشد که به تناوب نمایش داده می شود و این تصور ایجاد می شود که پاهای روح حرکت می کند.

[۸] اسپرایت های مورد استفاده از سایت https://tcrf.net/Pac-Man_%28Arcade%29 استخراج شده اند.



شکل ۴-۳ اسپرایت های مورد استفاده برای روح های مختلف

شکل ۴-۴ اسپرایت مورد استفاده برای روح های ترسیده



شکل ۴-۵ اسپرایت مورد استفاده برای میوه ها

برای ایجاد انیمیشن پک-من کد زیر در حلقه اصلی بازی نوشته شده است.

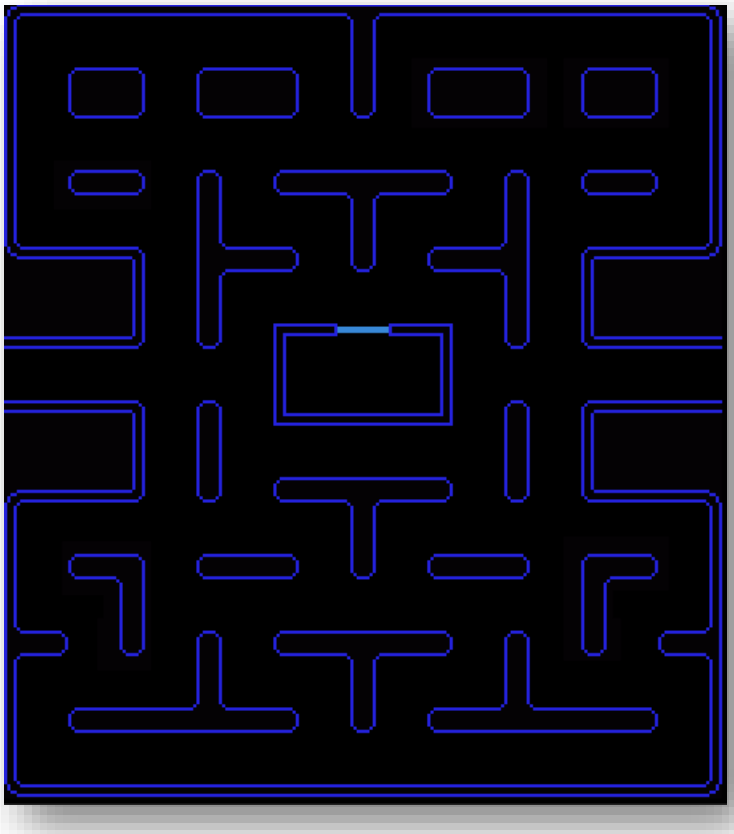
```
Time deltatime = clock.restart();
zamaneseparishode += deltatime;
float zamanbarhasbesanieh = zamaneseparishode.asSeconds();

int frame = static_cast<int>((zamanbarhasbesanieh / zamaneanimation) * tedadFrame) % tedadFrame;
PacMan.setTextureRect(IntRect(frame * andaze.x, 0, andaze.x, andaze.y));
```

متغیر deltatime زمان هر بار اجرای حلقه بازی را در خورد ذخیره می کند. سپس کل زمان سپری شده بر حسب ثانیه را محاسبه می نماییم. فرض کرده ایم که زمان اجرای انیمیشن پک-من ۰,۲ ثانیه باشد و می دانیم انیمیشن پک من از ۳ فریم تشکیل شده است. اکنون با تقسیم زمان سپری شده از بازی بر زمان انیمیشن (که ۰,۲ می باشد) تعداد انیمیشن های کامل به دست می آید و با ضرب در تعداد فریم (که ۳ می باشد) ، تعداد کل فریم هایی که تا این لحظه پخش شده است محاسبه می شود. اکنون اگر باقی مانده این مقدار را بر تعداد فریم(که ۳ می باشد) به دست بیاوریم متوجه می شویم که در این لحظه کدام فریم باید نمایش داده شود. سپس توسط تابع setTextureRect بخشی از شکل ۴-۱ که مورد نیاز باید را برش داده و نمایش می دهیم.

۴ – ۳ نقشه بازی

برای نقشه بازی از نگاشت یک ماتریس بر روی یک عکس استفاده شده است. عکس مورد استفاده در بخش زیر قابل مشاهده است.[۹]

[۹] عکس مورد استفاده برای نقشه عرض ۴۵۱ پیکسل و ارتفاع ۵۰۰ پیکسل دارد. برای نگاشت بهتر، مقیاس این عکس در پروژه کمی افزایش یافته است.



شکل ۴-۶ عکس مورد استفاده برای نقشه بازی

یک ماتریس ۱۹ * ۲۲ (۲۲ سطر و ۱۹ ستون) به عنوان نقشه بازی در نظر گرفته شده است. سپس این ماتریس بر روی شکل ۴-۶ نگاشت شده است تا گرافیک بازی شکل بگیرد. محتوای خانه ها با اعداد نمایش داده شده در جدول ۴-۱ مطابقت دارد.

عدد	معنی
۰	خوراک معمولی
۱	خوراک قدرتی
۲	دیوار
۳	خالی (یعنی این خانه یا خوراکی بوده که خورده شده یا از ابتدا خالی بوده است)
۵	درِ خانه ارواح

جدول ۴-۱ مقادیر مورد استفاده در جدول نقشه

برای نگاشت ماتریس بر روی نقشه بازی از محاسبات ریاضی استفاده شده است. به عنوان مثال:

```
map[(int)ceil((y - 200) / 24.9)][(int)ceil((x - 20) / 25)]
```

نقشه بازی از نقطه (۲۰۰و۲۰) ترسیم شده است. در نتیجه زمانی که بخواهیم نگاشت از مختصات جسمی بر روی ماتریس داشته باشیم باید از y آن نقطه ۲۰۰ واحد و از x آن نقطه ۲۰ واحد کم کنیم. همچنین در زمان ترسیم نقشه، برای نگاشت بهتر نقاط بر روی تصویر نقشه بین هر ستون در نقشه بازی ۲۵ پیکسل و بین هر سطر ۲۴,۹ پیسکل فاصله در نظر گرفته شده است که برای محاسبه نگاشت نقشه به ماتریس و برعکس باید به آن دقت نمود.

۴ – ۴ بنر بازی

بنر یک عکس در بالای محیط بازی می باشد که با کلیک بر روی آن دستور العمل بازی نمایش داده می شود.

شکل ۴-۷ بنر بازی پک من

شکل ۴-۸ دستورالعمل بازی در زمان کلیک بر روی بنر

۴ – ۵ منوی بازی

برای راحتی کاربر در زمان اجرای بازی یک منوی ساده در گوشه پایین و سمت راست بازی در نظر گرفته شده است.

شکل ۴-۹ منوی بازی

این منو از دو طریق فعال می شود. حالت اول زمانی است که روی منو کلیک می کنیم در نتیجه این منو فعال شده و عملکرد مورد نظر را انجام می دهد. ابتدا توسط SFML کلیک موس را تشخیص می دهیم. سپس بررسی می کنیم که موقعیت کلیک در محدوده هر منویی هست یا خیر؟ اگر در محدوده باشد عملیات مورد نظر انجام می شود.

با کلیک روی منوی Clear High Score فایل ثبت بیشترین امتیاز باز شده و امتیاز ۰ در آن ذخیره می شود. با کلیک روی منوی Exit بازی بسته می شود. با کلیک روی منوی Reset بازی به حالت اولیه رفته و مجدد آماده شروع می شود. در قطعه کد زیر نحوه نگارش بخشی از کد قابل رویت می باشد.[۱۰]

```
if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
{
auto mouse_pos = sf::Mouse::getPosition(MainWindow);
auto translated_pos = MainWindow.mapPixelToCoords(mouse_pos);
if
(ClearHighScoreMenu.getGlobalBounds().contains(translated_pos))
{
     ofstream output("Data/HighScore.txt", ios::out);
     output << 0;
     output.close();
}
if (ExitMenu.getGlobalBounds().contains(translated_pos))
{
      MainWindow.close();
}
```

[۱۰] جهت سهولت و خوانایی کد، از درج توضیحاتی که در VSCode نوشته شده اند خودداری کرده ایم. اما توضیحات خط به خط در فایل پروژه موجود می باشد

```
if (banner.getGlobalBounds().contains(translated_pos))
{
     NemeyesheDastoreBazi = !NemeyesheDastoreBazi;
}
```

حالت دوم استفاده از کلیدهای میانبری است که برای منوها در نظر گرفته شده است. در این حالت با فشردن کلید Q عملیات ریست ، کلید C عملیات پاک کردن بیشترین امتیاز و کلید Esc بازی بسته می شود.

به عنوان مثال قطعه کد زیر پاک کردن بیشترین امتیاز را نمایش می دهد.

```
if (event.key.code == Keyboard::Key::C)
   {
     ofstream output("Data/HighScore.txt", ios::out);
     output << 0;
     output.close();
    }
```

۵ توابع مورد استفاده

در این بخش بر روی توابعی که برای هدفی خاص طراحی شده اند بحث خواهیم کرد.

۵ – ۱ تابع Wincheck

این تابع در برنامه اصلی نوشته شده است و وظیفه بررسی شرایط برد را بر عهده دارد. بدیهی است که برد زمانی رخ می دهد که تمام خوراکی های معمولی خورده شده باشد. پس از خوردن یک خوراکی عادی همواره این تابع فراخوانی می شود تا شرایط برد بررسی شود. پارامتر این تابع نقشه بازی می باشد که یک ماتریس ۱۹ * ۲۲ می باشد و یک مقدار false/true بر می گرداند.

```
bool winCheck(int map[22][19])
{
    for (size_t i = 0; i < 22; i++)
    {
        for (size_t j = 0; j < 19; j++)
        {
            if (map[i][j] == 0)
            {
                return false;
            }
        }
    }
    return true;
}
```

۵ – ۲ تابع setmap

این تابع برای چینش نقشه مورد استفاده قرار می گیرد. محل قرارگیری دیوارها و خوراکی های عادی و قدرتی در این تابع مشخص می شود. برای پر کردن نقشه بازی از جدول ۴-۱ استفاده شده است.

در این تابع ابتدا، تمام نقاط به ۰(خوراکی عادی) مقدار دهی می شود، سپس دیوارها و نقاط قدرتی و موقعیت های خالی مشخص می شود.

```
    for (size_t i = 0; i < 22; i++)
    {
```

```
        for (size_t j = 0; j < 19; j++)
        {
            map[i][j] = 0;
        }
    }
```

در قطعه کد زیر دیوار سمت بالای نقشه مشخص شده است.

```
for (size_t i = 0; i < 19; i++)
    {
        map[0][i] = 2;
    }
```

در قطعه کد زیر نیر یک خانه از نقشه به عنوان دیوار مشخص شده است.

```
map[1][9] = 2;
```

۵ – ۳تابع drawmap

این تابع دو پارامتر دریافت می کند، پارامتر اول نام پنجره ای که ترسیمات بر روی آن انجام می شود را دریافت می کند و پارامتر دوم یک ماتریس ۱۹*۲۲ می باشد که نقشه بازی را تداعی می کند. بخشی از کد که خوراکی های عادی را ترسیم می کند را در قسمت زیر مشاهده می کنیم. برای ترسیم خوراکی های عادی از یک دایره زرد رنگ به شعاع ۲ پیکسل استفاده شده است.

```
 if (map[i][j] == 0)
{
   CircleShape c(2);
   c.setFillColor(Color::Yellow);
   c.setPosition(Vector2f(x, y));
   b.draw(c);
}
```

برای ترسیم خوراکی های قدرتی نیز از یک دایره سفید توپر به شعاع ۵ پیکسل استفاده شده است.

```
else if (map[i][j] == 1)
```

```
{
   CircleShape c(5);
   c.setFillColor(Color::White);
   c.setPosition(Vector2f(x, y));
   b.draw(c);
}
```

دیوارها در زمان ترسیم مورد استفاده قرار نمی گیرند اما برای جلوگیری از حرکت روح ها و پک-من استفاده می شوند. اگر دیوار ها با مستطیلی به عرض و ارتفاع ۱۰ ترسیم شوند محیط بازی به شکل زیر خواهد بود.

شکل ۵-۱ ترسیم بازی همراه با دیوارها

این دیوارها در زمان بازی قابل رویت نیستند و برای زیبا تر شدن محیط بازی از ترسیم آن ها جلوگیری شده است.

در این تابع بین هر دو ستون ۲۵ پیکسل و بین هر دو سطر ۲۴,۹ پیکسل فاصله می باشد. این اعداد از این جهت در نظر گرفته شده اند که با این اعداد انطباق بیشتری در نگاشت ماتریس به نقشه بازی اتفاق می افتد.

۶ کلاس های مورد استفاده

۶ – ۱ کلاس ghost

این کلاس برای مشخص کردن روح ها در بازی مورد استفاده قرار می گیرد. ویژگی های مورد استفاده برای توصیف یک روح به شرح زیر می باشد:

- ✓ مختصات افقی و عمودی : مختصات افقی و عمودی برای تعیین موقعیت روح در نقشه بازی مورد استفاده قرار می گیرد.
- ✓ اسپرایت برای حالت عادی ، ترسیده و چشم : برای وضعیت های مختلف روح (عادی – ترسیده – خورده شده) نیز اسپرایت های مختلف مورد استفاده قرار گرفته است.
- ✓ سرعت : حالت های مختلف روح سرعت های مختلفی دارد. برای توصیف سرعت حرکت روح از این متغیر استفاده می شود.
- ✓ جهت حرکت : جهت حرکت روح را مشخص می نماید. در جدول ۶-۱ مقادیر درنظر گرفته شده مشخص شده است.

عدد	جهت حرکت
۱	حرکت به سمت راست
۲	حرکت به سمت پایین
۳	حرکت به سمت چپ
۴	حرکت به سمت بالا

جدول ۶-۱ جهت حرکت

- ✓ وضعیت : یک روح در چند وضعیت می تواند قرار بگیرد که توسط یکی از اعداد جدول ۶-۲ مشخص می شود.

عدد	وضعیت
۱	روح در حالت پرسه زنی
۲	روح در حالت تعقیب
۳	روح در حالت ترسیده و رنگ آبی
۴	روح در حالت ترسیده و رنگ سفید
۵	روح خورده شده

جدول ۶-۲ وضعیت روح

۶ – ۲ تابع getStatus

برای دریافت وضعیت روح مورد استفاده قرار می گیرد.

۶ – ۳ تابع setStatus

برای تعیین وضعیت روح مورد استفاده قرار می گیرد.

۶ – ۴ تابع draw

برای ترسیم اسپرایت روح بر روی نقشه بازی مورد استفاده قرار می گیرد. با توجه به وضعیت روح یکی از تصاویر را برای روح در نظر گرفته و ترسیم می کند. پارامتر اول پنجره ای است که ترسیم اشکال بر روی آن انجام می شود.

```
void Ghost::draw(sf::RenderTarget &target, sf::RenderStates states) const
{
    if (status == 3 || status == 4)
    {
        target.draw(sSprite, states);
    }
    else
        target.draw(mSprite, states);
}
```

۶ – ۵ تابع setPosition

این تابع برای حرکت روح به مختصات مشخص مورد استفاده قرار می گیرد. دو پارامتر برای مختصات افقی و عمودی دریافت نموده و به مختصات روح اختصاص می دهد.

۶ – ۶ تابع getGlobalBounds

از این تابع برای دریافت مستطیل محاط بر روح استفاده می شود. این تابع مستطیل محاط بر روح رو به عنوان نتیجه بر می گرداند که برای تشخیص برخورد روح با سایر اجسام مورد استفاده قرار می گیرد.

برای تشخیص برخورد دو شی در SFML کافی است تا برخورد مستطیل های محاط بر دو شی را بررسی نماییم. به عنوان مثال در کد زیر بر خورد پک-من با روح قرمز بررسی شده است. این برخورد از دو حالت خارج نیست. یا روح در حالت ترسیده بوده و پس از برخورد به دو چشم تبدیل شده و به خانه برمی گردد یا اینکه در حالت پرسه زنی یا تعقیب بوده و یک جان از پک-من کم می شود.

```
if (PacMan.getGlobalBounds().intersects(redGhost.getGlobalBounds()))
{
   redGhost.setStatus(1);
   // redGhost.setPosition(245, 375);
   redGhost.boroBe(map);
}
```

در کد فوق ابتدا مستطیل محاط بر پک-من محاسبه شده است، سپس توسط تابع intersects برخورد آن با مستطیل محاط بر روح قرمز مورد بررسی قرار گرفته است.

۶ – ۷ تابع ShoroeTars

در این تابع فرآیند تبدیل وضعیت روح از حالت عادی به وضعیت ترس شکل می گیرد. ابتدا وضعیت روح به حالت ترسیده تبدیل می شود سپس جهت خود را ۱۸۰ درجه تغییر می دهد.

```
status = 3;
if (jahateHarekat == 1)
{
   jahateHarekat = 3;
}
else if (jahateHarekat == 2)
{
    jahateHarekat = 4;
}
else if (jahateHarekat == 3)
{
    jahateHarekat = 1;
}
else if (jahateHarekat == 4)
{
    jahateHarekat = 2;
}
```

۶ – ۸ تابع boroBe

در این تابع در زمانی که به یک تقاطع برسیم تصمیم گیری می کنیم که کدام مسیربه هدف ما نزدیک تر است. به عنوان مثال زمانی که یک روح در وضعیت تعقیب قرار دارد هدف مدنظر روح رسیدن به پک-من می باشد. در نتیجه زمانی که به تقاطع می رسد تصمیم گیری می کند که کدام مسیر برای رسیدن به روح نزدیک تر است.

در کد زیر بررسی شده که به نقطه ۹و۹ [11]چقدر نزدیک هستیم و برای رسیدن به آن محاسبات را انجام داده ایم.

```
sorat = 0.1;
float xDist = x - 9 * 25 + 20;
float yDist = y - 9 * 24.9 + 200;
float dist = sqrt((xDist * xDist) + (yDist * yDist));

if ((int)ceil((y - 200) * 24.9) < 11 && (int)ceil((x - 20) * 25) < 9)
{
    x += xDist * sorat;
    y += yDist * sorat;
 }
```

۶ – ۹ تابع masirhayeMojaver

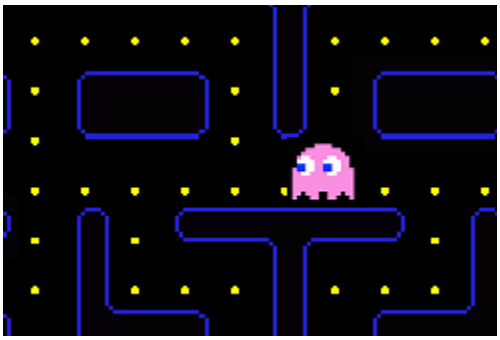
از این تابع برای پیدا کردن مسیر های ممکن برای حرکت در هر لحظه مورد استفاده قرار می گیرد. این تابع یک عدد را به عنوان خروجی برمیگرداند که نشان دهنده مسیرهای باز برای روح در چهار جهت آن می باشد. عددی که این تابع به عنوان خروجی برمیگرداند یک عدد مابین ۱ تا ۱۵ می باشد که مطابق جدول زیر محاسبه می شود.

عدد	جهت
۱	اگر راه سمت بالای روح باز باشد
۲	اگر راه سمت چپ روح باز باشد
۴	اگر راه به سمت پایین روح باز باشد
۸	اگر راه به سمت راست روح باز باشد

جدول ۶-۳ مقادیر مورد استفاده جهت بررسی مسیر های ممکن برای حرکت روح

[11] این نقطه مختصات درب خروجی خانه ارواح می باشد.

به عنوان مثال اگر روح در وضعیت شکل ۶-۱ باشد (یعنی بر سر یک سه راهی قرار گرفته باشد) تابع masirhayeMojaver برای آن عدد ۱۱ را بر می گرداند.



شکل ۶-۱ روح قرار گرفته در موضعیت یک سه راهی

با توجه به شکل متوجه می شویم راه سمت راست – چپ و بالا برای روح باز هستند.در نتیجه خروجی تابع مجموع ۱+۴+۸ می شود که معادل عدد ۱۱ می باشد. از خروجی این تابع در توابع دیگر استفاده می شودو با توجه به خروجی این تابع تصمیم گیری برای ادامه حرکت انجام می شود. زمانیکه روح در وضعیت تعقیب باشد باید از روی خروجی این تابع مناسب ترین مسیر را انتخاب کند یا زمانی که در حالت پرسه زنی می باشد بر اساس این تابع تصمیم می گیرد که به شکل تصادفی بر روی کدام مسیر پرسه بزند!

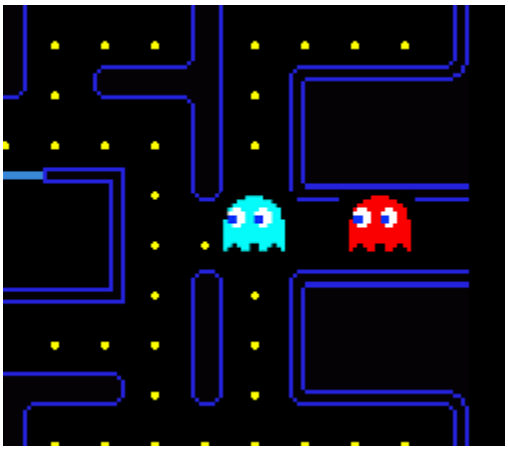
```
int Ghost::masirhayeMojaver(int map[22][19])
{
int count = 0;
if (map[(int)(ceil((y - 200) / 24.9) - 1)][(int)ceil((x - 20) / 25)] != 2)
{
  count += 1;
}
if(map[(int)(ceil((y - 200) / 24.9))][(int)(ceil((x - 20) / 25)- 1)] != 2)
{
  count += 2;
}
if (map[(int)(ceil((y - 200) / 24.9) + 1)][(int)ceil((x - 20) / 25)] != 2
&& map[(int)(ceil((y - 200) / 24.9) + 1)][(int)ceil((x - 20) / 25)] != 5)
{
  count += 4;
}
if(map[(int)(ceil((y - 200) / 24.9))][(int)(ceil((x - 20) / 25) + 1)]!= 2)
```

```
{
   count += 8;
}
return count;
}
```

همانطور که در کد مشاهده می شود ابتدا نگاشت مختصات به نقشه انجام می شود و در صورتی که در سمت مورد نظر دیوار وجود نداشته باشد مقادیر متناظر جدول ۶-۳ با متغیر count جمع می شود و در نهایت به عنوان خروجی برگردانده می شود.

۶ – ۱۰ تابع masireShansi

این تابع بر اساس وضعیت روح و خروجی تابع masirhayeMojaver[۱۲] اقدام به تصمیم گیری می کند تا مسیر بعدی روح را مشخص نماید. اگر روح در وضعیت تعقیب باشد، زمانی که به تقاطع می رسد تصمیم گیری می کند که کدام مسیر برای رسیدن به پک-من مناسب تر می باشد و به همان مسیر می رود. اگر روح در وضعیت پرسه زنی باشد نیز در تقاطع ها به شکل شانسی یک مسیر را انتخاب کرده و ادامه می دهد. البته طبق قانون موجود در بازی پک-من باید جهتِ ورود به تقاطع را بدانیم تا زمانی که می خواهیم تصمیم گیری برای ادامه حرکت شانسی بگیریم حرکت خلاف جهت ورود (۱۸۰ درجه) انجام ندهیم.



شکل ۶-۲ روح فیروزه ای بر سر چهارراه تصمیم گیری تصادفی قرار گرفته است

کد تصمیم گیری تصادفی بر سر یک چهار راه را قطعه کد زیر مشاهده می کنیم.

[۱۲] که عددی مابین ۱ و ۱۵ می باشد

```
else if (count == 15)
{
  tas = (rand() % 3) + 1;
  if (jahateHarekat == 1)
  {
     if (tas == 1)
     {
        return 4;
     }
     else if (tas == 2)
     {
         return 1;
     }
     else
         return 2;
  }
   else if (jahateHarekat == 4)
   {
     if (tas == 1)
     {
         return 3;
     }
     else if (tas == 2)
      {
           return 1;
      }
      else
           return 4;
  }
  else if (jahateHarekat == 3)
  {
      if (tas == 1)
      {
           return 4;
      }
```

```
        else if (tas == 2)
        {
            return 3;
        }
        else
            return 2;
    }
    else if (jahateHarekat == 2)
    {
        if (tas == 1)
        {
            return 3;
        }
        else if (tas == 2)
        {
            return 2;
        }
        else
            return 1;
    }
}
```

در این قطعه کد ابتدا یک عدد تصادفی بین ۱ تا ۳ تولید می شود. چون نمی خواهیم مسیری که آمده ایم را برگردیم، در نتیجه جهت ورود به تقاطع را بررسی می کنیم. به عنوان مثال اگر جهت حرکت ۱ باشد، براساس جدول ۶-۱ در حال حرکت به راست بوده ایم، در نتیجه روح نباید حرکت در خلاف جهت این حرکت (حرکت به سمت چپ) داشته باشد و در نتیجه در تصمیم گیری حرکت بعدی مورد محاسبه قرار نمی گیرد.

اگر روح به سمت راست در حال حرکت باشد بین جهت های بالا ، پایین و راست براساس متغیر تصادفی tas تصمیم گیری می شود. اگر متغیر tas برابر ۱ باشد روح به سمت بالا می رود، در غیراینصورت اگر متغیر tas برابر ۲ باشد روح به سمت راست می رود و حرکت قبلی را ادامه می دهد، در غیراینصورت به سمت پایین حرکت خواهد نمود.

۶ – ۱۱ تابع taeineJahat

از این تابع برای تعیین جهت روح و در نتیجه مسیر بعدی حرکت روح استفاده شده است. در این تابع ابتدا ورود به مناطق خاص بررسی می شود و با توجه به آن منطقه برای مسیر بعدی تصمیم گیری می شود. به عنوان مثال اگر روحی به منطقه تونل جادویی رسیده باشد باید از سردیگر تونل وارد شود.نمونه کد زیر به بررسی رسیدن به منطقه تونل جادویی و عبور از آن می پردازد.

```
if ((int)ceil((x - 20) / 25) == 0 && (int)ceil((y - 200) / 24.9) == 10 &&
jahateHarekat == 3)
{
    x = 20 + 18 * 25;
}
if ((int)ceil((x - 20) / 25) == 18 && (int)ceil((y - 200) / 24.9) == 10 &&
jahateHarekat == 1)
{
    x = 20;
}
```

در این تابع ابتدا مختصات روح به نقشه بازی نگاشت می شود و در صورتی که روح در دهانه تونل قرار گرفته باشد در دهانه دیگر ظاهر می شود.

در صورتی که جهت حرکت روح به سمت راست باشد و هنوز به دیوار نرسیده باشد باید به حرکت خود ادامه دهد. اما اگر به دیوار برسد بررسی می کند که چه جهاتی از مسیر های ممکن باز است و مسیر بعدی را از بین مسیرهای ممکن انتخاب می کند.

```
if (jahateHarekat == 1)
{
  if (map[(int)ceil((y - 200) / 24.9)][(int)ceil((x - 20 + (10 * sorat)) / 25)] != 2)
  {
     x += (0.1 * sorat);
    jahateHarekat = masireShansi(map);
  }
  else
  {
     jahateHarekat = masireShansi(map);
  }
}
```

۶ – ۱۲ تابع masafat

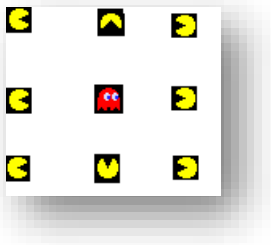
این تابع مسافت مختصات x و y روح و پک-من را محاسبه کرده و به عنوان نتیجه یک بردار حاوی اختلاف مسافت روح و پک-من بر می گرداند. کد این تابع در قطعه کد زیر قابل مشاهده است.

```
    float distX = x - pacmanX;
    float distY = y - pacmanY;
    return Vector2f(distX, distY);
```

۶ – ۱۳ تابع Taaghib

این تابع برای تعقیب پک-من مورد استفاده قرار گرفته است. این تابع در زمان قرار گرفتن بر روی تقاطع ها مسیر جدید را بر اساس معیار نزدیکی به پک-من مورد بررسی قرار می دهد و سعی می کند به هدف نزدیک شود. اساس محاسبات این تابع ، تابع masafat می باشد که بخش قبل مورد بررسی قرار گرفت.

در هر تقاطع بر اساس راه هایی که برای روح باز است تصمیم گیری انجام می شود. در هر تقاطع برای اینکه جامعیت در مسیر یابی وجود داشته باشد حالت های زیر مورد بررسی قرار گرفته است.



شکل ۶-۳ انواع حالت های قرار گیری پک-من نسبت به روح

در هر تقاطع بر اساس راه هایی ک وجود دارد و حالت هایی که روح نسبت به پک من دارد تصمیم گیری انجام گرفته است. نمونه از کد تصمیم گیری در قطعه کد زیر آورده شده است.

```
Vector2f dist = masafat(pacmanX, pacmanY);
if (abs(dist.x) < 0.5 && dist.y > 0)
{
    return 4;
}
else if (abs(dist.x) < 0.5 && dist.y < 0)
```

```
{
    return 2;
}
else if (dist.x > 0 && abs(dist.y) < 0.5)
{
    return 3;
}
else if (dist.x < 0 && abs(dist.y) < 0.5)
{
    tas = rand() % 2;
    if (tas == 0)
    {
       return 4;
    }
    else
       return 2;
}
…….
```

۷ کدهای خاص

۷ – ۱ بیشترین امتیاز

برای ثبت بیشترین امتیاز از یک فایل استفاده شده است. زمانی که بازی جدید شروع می شود ابتدا بیشترین امتیاز از فایل خوانده شده و بر روی پنجره و قسمت در نظر گرفته شده برای آن نمایش داده می شود.

شکل ۷-۱ محل نمایش بیشترین امتیاز

کد خواندن از فایل در قطعه کد زیر قابل رویت می باشد:

```
    ifstream inputfile("Data/HighScore.txt", ios::in);
    int highScoreFromFile = 0;
    inputfile >> highScoreFromFile;
```

در برخی زمان ها نیاز به بررسی لزوم ثبت بیشترین امتیاز در فایل می باشد. اگر بازی را ببریم یا ببازیم باید بررسی کنیم که آیا امتیاز به دست آمده از بیشترین امتیاز بالاتر است یا خیر؟ در صورت لزوم باید این اطلاعات بر روی فایل به روز رسانی شود. قطعه کد زیر فرایند بررسی را نمایش می دهد:

```
if (emtiaz > highScoreFromFile)
{
 ofstream output("Data/HighScore.txt", ios::out);
 output << emtiaz;
 output.close();
 highScore.setString("High Score: " + std::to_string(emtiaz));
}
```

این شرط بررسی می کند که اگر امتیاز بازی خاتمه یافته از بیشترین امتیاز ثبت شده در فایل بیشتر باشد ، در نتیجه بیشترین امتیاز جدیدی به ثبت رسیده است که باید فایل را به روز رسانی کنیم.

۸ چالش ها

۸ – ۱ یافتن مسیر های باز

یکی از اساسی ترین چالش های این پروژه یافتن مسیرهای باز روح ها در هر لحظه بود. برای غلبه بر این چالش از ویژگی اعداد مبنای ۲ استفاده کردیم.[۱۳]می دانیم اعداد با استفاده از توان های ۲ فقط با یک شیوه نمایش ساخته می شوند، به عنوان مثال ۷ تنها می تواند با جمع ۱و۲و۴ ایجاد شود. یا ۱۵ تنها می تواند با جمع ۱و۲و۴و۸ ساخته شود و راه دیگری برای ایجاد ۱۵ با ترکیب دیگری[۱۴]از توان های ۲ وجود ندارد.

در نتیجه در هر مرحله بررسی مسیرهای باز برای یک روح ، تابع masirhayeMojaver یک عدد به عنوان خروجی بر می گرداند که از آن عدد متوجه مسیرهای باز مجاور می شویم و می توانیم برای حرکت های بعدی آن مسیرها را در نظر بگیریم.

۸ – ۲ درب خانه ارواح

ابتدای بازی ارواح قادر به عبور از درب می باشند اما دیگر ارواح و پک-من قادر نخواهند بود ازآن داخل خانه شوند. یکی از چالش ها جلوگیری از ارواح برای ورود مجدد به داخل خانه بود.

برای حل این معزل نوع درب خانه ارواح با شماره ۵ در ماتریس نقشه بازی مقدار دهی شده است. درنتیجه با این مقداردهی جدید می توان بررسی کرد اگر ارواح به سمت بالا در حال حرکت بودند قادر به عبور از درب می باشند ، اما اگر ارواح یا پک–من به سمت پایین حرکت کنند باید بررسی شود تا درب خانه در پایین نباشد و گرنه باید از حرکت ارواح یا پک-من جلوگیری نمود

۸ – ۳ نگاشت نقشه به ماتریس

همانطور که در بخش های قبلی ذکر شد، برای نمایش نقشه بازی از یک ماتریس ۱۹ * ۲۲ استفاده شده است. چالش اصلی نگاشت ماتریس بر روی نقشه بازی بود در حالی که بیشترین انطباق بین نقشه و ماتریس شکل بگیرد. برای غلبه بر این چالش، نقشه بازی را مقیاس[۱۵]دهی کردیم و همچنین برای نمایش ماتریس بین هر دو ستون ۲۵ پیکسل و بین هر دو سطر ۲۴,۹ پیکسل فاصله وجود دارد. در نتیجه با این مقادیر بیشترین انطباق بین ماتریس و نقشه بازی شکل می گیرد.

```
map[(int)ceil((y - 200) / 24.9)][(int)ceil((x - 20) / 25)]
```

[۱۳] در جدول ۳-۶ مقادیر مختلف برای نمایش مسیر باز قابل مشاهده است.

[۱۴] ترکیب غیر تکراری

[۱۵] scale

دلیل کسر ۲۰ از x و ۲۰۰ از y نیز شروع ترسیم نقشه از مختصات (۲۰۰و۲۰) می باشد.

۸ – ۴ تعقیب و پرسه

روح ها باید بر اساس یک زمان بندی مشخص بین حالت های پرسه زدن و تعقیب تغییر حالت پیدا کنند. برای مشخص نمودن رفتار پیوسته بین حالت های پرسه و تعقیب از یک آرایه استفاده شده است. در این آرایه زمان های پرسه و تعقیب به ترتیب نوشته شده است.

```
int traceTimer[] = {0, 7, 20, 7, 20, 5, 20, 5, 20};
```

پس از اینکه آخرین تعقیب انجام شود(آخرین ۲۰ ثانیه) مجددا محاسبات تعقیب و پرسه از ابتدای آرایه شروع می شود یا به عبارتی دیگر تا بی نهایت ادامه می یابد!

ابتدا بازی بر اساس زمان تنظیم شده در آرایه تایمر کار می کند(طبق کد زیر) و ۷ ثانیه ارواح پرسه می زنند، سپس تغییر وضعیت داده و ۲۰ ثانیه به تعقیب پک-من می پردازند و به همین ترتیب تا انتهای بازی ادامه می دهند.

```
ghostsMoveTimer += deltatime.asSeconds();
if (ghostsMoveTimer >= traceTimer[traceTimerPicker])
{
   int tempStatus = (((traceTimerPicker) % 2) + 1);
   traceTimerPicker = traceTimerPicker % 9 + 1;
   ghostsMoveTimer = 0;
   if (redGhost.getStatus() != 5)
       redGhost.setStatus(tempStatus);
   if (cyanGhost.getStatus() != 5)
       cyanGhost.setStatus(tempStatus);
   if (orangeGhost.getStatus() != 5)
        orangeGhost.setStatus(tempStatus);
   if (pinkGhost.getStatus() != 5)
       pinkGhost.setStatus(tempStatus);
 }
```

متغیر کمکی tempStatus باعث می شود تا همواره مقدار ۱ یا ۲ داشته باشیم که نشان دهنده وضعیت در حال تعقیب و در حال پرسه زدن روح می باشد (طبق جدول ۶-۲). متغیر traceTimerPicker نیز کمک می کند در

هر زمان تایمر یکی از خانه های آرایه را انتخاب کند و با تقسیم بر ۹ شدن امکان بی نهایت بار تکرار این فرایند را می دهد.

برآمدهای آموزشی

- ✓ آشنایی با پیاده سازی انیمیشن دو بعدی توسط اسپرایت
- ✓ اندازه گیری زمان اجرا
- ✓ حرکت دادن اشیاء روی صفحه
- ✓ ترسیم اشیای گرافیکی بر روی صفحه
- ✓ آشنایی با مفاهیم رویداد در برنامه نویسی
- ✓ آشنایی با سیستم مختصات
- ✓ آشنایی با مفهوم برخورد [۱۶]

[۱۶] collision

۹ منابع

- ✓ سایت https://www.sfml-dev.org/
- ✓ سایت https://stackoverflow.com/
- ✓ کتاب آموزش مقدماتی برنامه نویسی بازی با استفاده از SFML نوشته میلکو جی. میلچیو – ترجمه عباسعلی طهماسبی