به نامِ خدا

درسِ مبانیِ برنامه‌سازی

نیم‌سالِ اول ۹۷-۹۶

دانشکدهٔ مهندسیِ کامپیوتر
دانشگاهِ صنعتیِ شریف

مستند فاز یک پروژه

موعد تحویل ۲۵ آذر ۱۳۹۶

نویسندگان **علی احمدی تشنیزی، سید سجاد کاهانی، محمدامینِ سالارکیا**

ویراستاریِ فنی و آماده‌سازی **سید سجاد کاهانی، علی احمدی تشنیزی**

آماده‌سازیِ کتابخانه **سید سجاد کاهانی**

- ممکن است برخی از اصطلاحات یا نوع‌داده‌های (type) استفاده شده در این مستند، برای شما ناشناخته باشند. معمولاً در پانویس‌ها توضیحاتِ بیشتر را خواهید یافت و می‌توانید از دوستانِ خود یا دستیارانِ آموزشی بپرسید. مطمئن باشید چیزهای سختی نیستند، اغلب نام‌گذاریِ جدیدی از چیزهایی که می‌دانید هستند.
- شما می‌توانید، هرکدام از تابع‌هایی که پیاده‌سازی نکردید را در طولِ زمانِ فازِ یک یا پس از آن، کدِ آن را دریافت کنید و تنها نمرهٔ پیاده‌سازیِ آن تابع را نخواهید گرفت.
- این مستند هنوز نهایی نیست و کامل‌تر خواهد شد.

۱ معرفیِ فاز

فاز اول به حرکت پک‌من و تعامل اشیاء مختلف بازی با یکدیگر می‌پردازد. وظیفه شما در این فاز، پیاده‌سازی کامل برنامه نیست، بلکه بخش‌هایی از آن قبلا توسط تیم پروژه پیاده‌سازی شده است و شما باید قسمت‌هایی مشخص از برنامه را پیاده‌سازی کنید به طوری که با تعامل با بخش‌های دیگر، فرایند منطقی بازی پیش برود. به مرور و در فازهای بعد، پیاده‌سازی بخش‌های دیگر نیز توسط خود شما صورت خواهد گرفت.

۲ پیش‌نیازها

۱.۲ cmake

ابتدا با مراجعه به داکِ `cmake` آن را برروی سیستمِ خود نصب کنید.

۲.۲ sdl

`sdl` یک کتاب‌خانهٔ گرافیکی برروی زبانِ سی و سی++ است که از بسیاری از سیستم‌عامل‌ها پشتیبانی می‌کند. شما باید آن را مطابقِ دستورالعملِ زیر برروی سیستم‌عاملِ خود نصب کنید.

۱.۲.۲ اوبونتو

با اجرا کردنِ دستورِ زیر در ترمینال، کتاب‌خانهٔ `sdl` نصب خواهدشد.

```
sudo apt-get install libsdl2-gfx-dev
```

۲.۲.۲ مک

ابتدا brew و Command Line Tools را نصب کنید. سپس بااجرا کردنِ دستور زیر در ترمینال، این کتاب‌خانه نصب می‌شود.

```
brew install sdl2_gfx
```

۳.۲.۲ ویندوز (۶۴بیتی)

این کتاب‌خانه برای ویندوز در فایلِ ضمیمهٔ پروژه در پوشهٔ `windows` قرار گرفته‌است. کافی‌ست پوشهٔ `sdl` را به فولدرِ پروژه کپی کنید.

۳.۲ کتاب‌خانهٔ پروژه

کتاب‌خانهٔ `liblib.a` مربوط به گرافیکِ پروژه است که با زحماتِ شبانه‌روزیِ «ستادِ یاری‌رسانی در سَمبَل کردن و جمع‌وجورکردنِ اموراتِ درس که اصن هیچ‌کس انجامشون نمی‌ده این‌روزا» آماده‌شده است.
این کتاب‌خانه در سه نسخهٔ `linux` و `mac` و `windows` تنها مخصوصِ سیستم‌های ۶۴بیتی در فولدرهای مذکور در فایلِ ضمیمهٔ پروژه قرار گرفته‌اند. فایلِ `liblib.a` مربوط به سیستم‌عاملِ خود را از فولدرِ مربوطه به فولدرِ پروژه کپی کنید.

۴.۲ شکلِ دقیق پروژه و Cmake

در اینجا نمودارِ درختیِ فایل‌های پروژه و شکلِ دقیق CmakeLists.txt را نشان میدهیم:

۱.۴.۲ لینوکس و مک

فولدرِ پروژه پس از اجرا کردنِ cmake و make کردنِ پروژه به شکلِ زیر خواهدبود.

```
project
├── cmake-build-debug
│   ├── res
│   │   └── map.txt
│   ├── project.o
│   └── ...
├── src
│   ├── game.c
│   ├── game.h
│   ├── input.h
│   ├── map.h
│   ├── physics.c
│   └── physics.h
├── CMakeLists.txt
└── liblib.a
```

فولدرِ res که فایلِ map.txt داخل آن است باید کنارِ خروجیِ اجراییِ پروژه باشد.
کد Cmake هم به صورتِ زیر است:

```
cmake_minimum_required(VERSION 3.8)
project(project)

set(CMAKE_C_STANDARD 99)
set(SOURCE src/physics.c src/game.c)
add_executable(project ${SOURCE})
include_directories(src)
add_library(lib STATIC IMPORTED)
set_target_properties(lib PROPERTIES IMPORTED_LOCATION "../liblib.a")
target_link_libraries(project lib)
target_link_libraries(project m SDL2 SDL2_gfx)
add_definitions(-D_REENTRANT)
```

دقت کنید در مثالِ بالا به جایِ project نامِ پروژهٔ خود را قرار دهید. ضمناً در خطِ اول که ورژنِ cmake version داده شده می‌توانید آن را عوض کنید و ورژنِ موردِ استفادهٔ خود را جایگزین کنید.

ویندوز

فولدرِ پروژه پس از اجرا کردنِ cmake و make کردنِ پروژه به شکلِ زیر خواهدبود.

```
project
├── cmake-build-debug
│   ├── res
│   │   └── map.txt
│   ├── project.exe
│   └── ...
├── sdl
│   ├── sdl2-gfx-lib
│   │   └── libsdl-gfx.a
│   └── sdl2-lib
│       ├── libSDL2.a
│       └── libSDL2main.a
├── src
│   ├── game.c
│   ├── game.h
│   ├── input.h
│   ├── map.h
│   ├── physics.c
│   └── physics.h
├── CMakeLists.txt
└── liblib.a
```

فولدرِ res که فایلِ map.txt داخل آن است باید کنارِ خروجیِ اجراییِ پروژه باشد.
کد Cmake هم به صورتِ زیر است

```
cmake_minimum_required(VERSION 3.6)
project(project)

set(CMAKE_C_STANDARD 99)
set(SDL2_GFX_LibDir "../sdl/sdl2-gfx-lib")
set(SDL2_Flags "-mwindows -Wl,--no-undefined -static-libgcc")
set(SDL2_LibDir   "../sdl/sdl2-lib")
add_library(SDL2     STATIC IMPORTED)
add_library(SDL2main STATIC IMPORTED)
add_library(SDL2_GFX STATIC IMPORTED)
set_property(TARGET SDL2     PROPERTY IMPORTED_LOCATION "${SDL2_LibDir}/libSDL2.a")
set_property(TARGET SDL2main PROPERTY IMPORTED_LOCATION "${SDL2_LibDir}/libSDL2main.a")
set_property(TARGET SDL2_GFX PROPERTY IMPORTED_LOCATION "${SDL2_GFX_LibDir}/libsdl-gfx.a"
    )
set(SDL2_Libs SDL2 SDL2main SDL2_GFX m dinput8 dxguid dxerr8 user32 gdi32 winmm imm32
    ole32 oleaut32 shell32 version uuid)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11 ${SDL2_Flags}")
set(SOURCE src/game.c src/physics.c)
add_executable(project ${SOURCE})
include_directories(src)
add_library(lib STATIC IMPORTED)
set_target_properties(lib PROPERTIES IMPORTED_LOCATION "../liblib.a")
target_link_libraries(project lib)
target_link_libraries(project ${SDL2_Libs})
```

دقت کنید در مثالِ بالا به جایِ project نامِ پروژهٔ خود را قرار دهید. ضمناً در خطِ اول که ورژنِ cmake version داده شده می‌توانید آن را عوض کنید و ورژنِ موردِ استفادهٔ خود را جایگزین کنید.

۵.۲ اجرا کردنِ پروژه

پس از انجام دادنِ مراحلِ بالا شما می‌توانید با استفاده از `cmake` و `make` همان‌طور که در مستندِ آموزشِ `cmake` توضیح داده‌شده پروژه را کامپایل کنید.

سپس می‌توانید فایلِ باینریِ آن را که در پوشهٔ `cmake-build-debug` قرار گرفته اجرا کنید.

نکته: توجه کنید که هنگامی که فایلِ باینریِ پروژه را از هرجایی اجرا می‌کنید در محلِ `res/map.txt` نسبت به آن‌جا فایلِ نقشه وجود داشته‌باشد.

ولی در وهلهٔ اول چیزی از پک‌من نمی‌بینید. (تنها یک‌صفحهٔ کوچک که چند نقطهٔ سفید روی پس‌زمینهٔ خاکستریِ آن وجود دارد) با پُر کردنِ تابع‌هایی که در فایل‌های `c.` وجود دارند، شما یک بازیِ پک‌من خواهید داشت.

۳ آماده‌سازیِ بازی

نقشهٔ بازی درونِ یک فایلِ متنی در فولدرِ res/map.txt ریخته‌شده‌است. شما باید تابعی را بنویسید که اطلاعات از فایلِ متنی گرفته و درونِ `struct` های مربوطه بریزد.

فرمت فایل تقریباً مشابهِ فازِ صفرم است. (تنها، زمان در آن وجود ندارد) به این صورت که در ابتدای فایل تعداد سطرها و تعداد ستون‌ها به صورت ۲ عدد صحیح آمده است. سپس در n سطر، هر یک به طول m ، هرخانه به صورت یک کاراکتر آمده است.

پس از آن در یک‌خط score بازی و در خط‌های بعدی به ترتیب اطلاعاتِ مربوط به پک‌من و روح‌ها می‌آیند.

مانندِ مثالِ زیر

```
5 9
#___**#**
###**###*
____****O
_##_^*#__
#_____#_*
27
pacman: 2 3 (2,0) (4,5)
blinky: 2 1 (3,7) (3,7)
pinky: 3 0 5 (3,7) (3,5)
clyde: 1 0 5 (3,7) (1,2)
inky: 2 0 20 (3,7) (3,8)
```

کاراکترهای متناظر با هر شئ در بازی نیز مشابهِ فازِ صفرم تعریف می‌شوند. کدِ زیر که قسمتی از فایلِ `map.h` است، این کاراکترها را به شکلِ ثوابتی تعریف کرده‌است که می‌توانید(بخوانید حتماً) از همین‌ها در کد زدن استفاده کنید. [۱]

```c
typedef enum {
    CELL_EMPTY= '_',
    CELL_BLOCK= '#',
    CELL_CHEESE= '*',
    CELL_CHERRY= '^',
    CELL_PINEAPPLE= 'O'
} Cell;
```

۱.۳ initiateGame (۲۱ امتیاز)

برای آماده‌سازیِ بازی، شما باید تابعی با عنوان initiateGame را پیاده‌سازی کنید که ساختار آن به شکل زیر است:

```c
void initiateGame(char* filename, Map* outMap, Game* outGame, Pacman* outPacman, Ghost*
    outGhosts)
```

تعریفِ این‌تابع در `game.h` و بدنهٔ آن در `game.c` خواهد بود.

هنگامِ فراخوانیِ این‌تابع، ابتدای برنامه خواهد بود و تنها یک‌بار فراخوانده‌می‌شود.

[۱] `enum` یک تایپ است (مثلِ `struct` و `int` و `char` و غیره که تنها می‌تواند مقادیرِ ثابتِ‌از پیش‌مشخص‌شده‌ای را به خود بگیرد. در تکه‌کدِ مذکور یک `enum` تعریف شده که تنها می‌تواند مقادیرِ مشخص‌شده را بگیرد و از این `enum` در `struct`ِ نقشه استفاده شده است.

۱.۱.۳ ورودی‌ها و خروجی‌ها

ورودی‌های این‌تابع به‌ترتیب `char* filename` یک رشته است که برابرِ آدرسِ فایلِ نقشه‌ای‌ست که باید فایلِ آن را بخوانید. `Map* outMap` یک اشاره‌گر به یک متغیر از نوعِ استراکتِ نقشه خواهدبود که باید آن‌را با داده‌های خواسته‌شده پر کنید.
`Map` که در فایلِ `map.h` تعریف شده‌است، اطلاعاتِ مربوط به طول و عرضِ نقشه و عناصرِ ثابتِ نقشه (تنها شاملِ دیوارها و انواعِ غذاها) در آن قرار می‌گیرد. [۲]

```
typedef struct {
    int width;
    int height;
    Cell cells[MAP_MAX_SIZE][MAP_MAX_SIZE];
} Map;
```

نکته: در داده‌های ذخیره‌شده در آرایهٔ `cells` در این استراکت، باید اندیسِ اول طول یا همان x یا ستون باشد و اندیسِ دوم عرض یا y یا سطر. احتمالاً این‌شیوه با چیزی که در قسمتِ قبل پیاده‌کرده‌اید متفاوت است.

ورودیِ بعدیِ این‌تابع `Game* outGame` است که یک اشاره‌گر به یک متغیر از جنسِ `Game` است که باید با اطلاعاتِ درست پر شود. این `struct` در `game.h` به صورتِ زیر تعریف‌شده‌است.

```
typedef struct {
    int score;
    int cheeses, cherries, pineapples;
    int ghosts;
} Game;
```

که در آن `score` امتیازِ کنونیِ بازی، `cheeses` تعدادِ پنیرهای موجود در نقشه و همین‌طور `cherries`، `pinapple` و `ghosts` به‌ترتیب تعدادِ گیلاس‌ها و آناناس‌ها و روح‌ها خواهند بود. (طبیعی‌ست که تعدادِ روح‌ها همواره برابرِ ۴ باشد.) لازم است همهٔ این مشخصه‌ها مقداردهی شوند و از پیش هیچ‌مقداری ندارند.

ورودیِ بعدی `outPacman Pacman*` است. که این نوعِ داده، به‌شکلِ زیر تعریف شده‌است.

```
typedef struct {
    double x, y;
    int startX, startY;
    Direction dir;
    int health;
    double speed;
} Pacman;
```

که در آن، `Direction` نیز نوعِ داده‌ایست که به شکلِ زیر تعریف می‌شود

```
typedef enum {
    DIR_UP = 1,
    DIR_RIGHT = 2,
    DIR_DOWN = 3,
    DIR_LEFT = 4,
    DIR_NONE = -1 } Direction;
```

در استراکتِ پک‌من `x` و `y` محلِ کنونیِ پک‌من است. این نقطه، دقیقاً گوشهٔ بالا-چپِ پک‌من را نشان می‌دهد. (توجه کنید که فرمتِ آن `double` است). `startX` و `startY` موقعیتِ اولیهٔ پک‌من است.

نکته: می‌دانیم که طول و عرضِ پک‌من و همهٔ‌اشیاءِ دیگرِ درونِ بازی برابر با ۱ است و همه را مربع فرض می‌کنیم.

`dir` جهتِ کنونیِ پک‌من است که یک‌متغیر از جنسِ `Direction` است که می‌توان آن را با یکی از علائمِ مشخص‌شده (شاملِ `{DIR_UP, DIR_RIGHT, DIR_LEFT, DIR_DOWN, DIR_NONE}`) یا اعدادِ متناظرِ آن‌ها پر کرد. همچنین می‌توان از این‌گونه‌متغیر به‌عنوانِ عدد در عبارات استفاده کرد. لازم به ذکر است که `DIR_NONE` به‌معنیِ ساکن بودن است.

همچنین `health` تعدادِ جان‌های پک‌من و `speed` سرعتِ پک‌من است که در ابتدا مقداری برابرِ ثابتِ `PACMAN_DEFAULT_SPEED` دارد که در `physics.h` تعریف شده‌است.

ورودیِ بعدی `outGhosts Ghost*` است که برخلافِ سه‌ورودیِ قبل یک **آرایه** از روح‌هاست و تضمین می‌شود که به تعدادِ کافی (`MAX_GHOST_COUNT`) خانه دارد.

---

[۲] گاهی در کدنویسی به‌جای تعریفِ `struct Map {...};` آن را به شکلِ `typedef struct {...} Map;` تعریف می‌کنند که در عمل، تنها فرقی که می‌کند این است که برای فراخوانیِ استراکت، به‌جای `struct Map` از `Map` استفاده می‌شود.
به‌طورِ دقیق‌تر اما `typedef A B` نوعِ `B` را با `A` هم‌ارز می‌کند که در این‌جا `B` همان `Map` است و `A` یک استراکتِ بدونِ نام است.

خانه‌های این‌آرایه نیز باید به ازای روح‌ها پُر شوند.
Ghost نیز به شکلِ زیر تعریف می‌شود

```
typedef enum { BLINKY, PINKY, CLYDE, INKY } GhostType;

#define CYCLES_PER_SEC 60

typedef struct {
    double x, y;
    int startX, startY;
    Direction dir;
    GhostType type;
    bool blue;
    unsigned long long blueCounterDown;
    double speed;
} Ghost;
```

در این ساختار x و y مشابهِ پک‌من مختصاتِ گوشهٔ بالا-چپِ روح است. startX و startY طول و عرضِ نقطهٔ شروع هستند و dir جهتِ حالِ حاضرِ روح است. type نیز متغیری از جنسِ GhostType است که می‌تواند یکی از مقادیرِ {BLINKY, PINKY, CLYDE, INKY} باشد.

blue آبی بودن یا نبودنِ روح را مشخص می‌کند. وقتی روح‌ها در حالتِ تدافعی قرار می‌گیرند، آبی می‌شوند و این متغیر برابرِ یک یا true قرار می‌گیرد و اگر روح در حالتِ تهاجمی باشد، مقدارِ این متغیر برابرِ با صفر خواهد بود. [۳] متغیرِ بعدی blueCounterDown زمانِ باقی‌مانده برای آبی بودن (تدافعی بودن) روح است. پس طبعاً مقدارِ این متغیر در حالتی که روح تهاجمی‌ست مهم نیست. ولی در حالتِ تدافعی باید برابر با تعدادِ cycle هایی باشد که روح تدافعی خواهد ماند.

ما هر ثانیه را به تعدادی cycle تقسیم کردیم که در هرکدام، پک‌من و روح‌ها مقدارِ کمی حرکت می‌کنند و نمایش داده می‌شوند و هربار وضعیتِ بازی چک می‌شود. برای توضیحِ دقیق‌تر می‌توان گفت که برنامه‌ای که نوشته‌شده یک‌حلقه دارد که در آن اشیاءِ بازی جابه‌جا می‌شوند، سپس رسم می‌شوند و سپس چک می‌شود امتیازها و وضعیتِ بازی چه تغییری کرده. این حلقه خیلی سریع اجرا می‌شود و در آن موجودات خیلی کم‌تر از یک خانه حرکت می‌کنند. این باعث می‌شود که به‌نظر بیاید پیوسته حرکت می‌کنند. به هربار اجرا شدنِ این حلقه، یک cycle می‌گوییم. هر CYCLES_PER_SEC تا cycle برابرِ یک ثانیه خواهد بود.

نکته: همان‌طور که گفته‌شد، ورودیِ برنامه، تعدادِ ثانیه‌هایی‌ست که روح در حالتِ تدافعی (آبی) است، ولی مقدارِ ذخیره‌شده در متغیرِ blueCounterDown باید تعدادِ cycleها باشد آخرین متغیر سرعتِ روح است که در شروعِ بازی باید برابر با ثابتِ GHOST_DEFAULT_SPEED باشد.

نکته: حتماً همهٔ این‌مشخصه‌هارا مقداردهی بکنید. مقدارهای اولیه معمولاً اعدادی تصادفی هستند و منجر به باگ‌های عجیب و سخت و پیدانشدنی‌ای می‌شوند

۴ حرکت‌ها

نکته: توجه کنید که پک‌من‌ها و روح‌ها می‌توانند از یک سمتِ نقشه خارج شوند و از سمتِ دیگر واردِ نقشه خواهند شد. قسمتِ عمده‌ای از این ویژگی در کتاب‌خانه پیاده‌سازی شده. شما تنها باید به برخورد با دیوارها و تعامل با دیگر اشیاءِ بازی توجه کنید.

۱.۴ decideGhost (۱۲ امتیاز)

این تابع، که بدنهٔ آن در physics.c وجود دارد، به شکلِ زیر تعریف شده‌است.

```
Direction decideGhost(const Map* map, Ghost* ghost)
```

این تابع وظیفه دارد که با توجه به وضعیتِ نقشه و مکانِ فعلیِ روح، جهتِ حرکتِ روح را مشخص کند. حرکتِ روح باید در این فاز به شکلِ کاتوره‌ای (random) باشد.

به این‌ترتیب که هرگاه روح دقیقاً در یک خانه قرار بگیرد (یعنی در مسیرِ بینِ دو خانه نباشد) باید جهتی را برای حرکت انتخاب کند این تابع فراخوانی می‌شود.

در لحظهٔ فراخوانیِ این‌تابع، مقادیرِ x و y که از نوعِ اعشاری هستند، اعدادِ صحیحی خواهند بود.

نکته: مقدارِ بازگشتیِ این‌تابع جهتِ حرکتِ روح را مشخص می‌کند و نباید هیچ‌تغییری در مشخصه‌های متغیرِ ghost داده‌شود. یعنی نباید جهت را روی روح اعمال کنیم.

اگر این‌تابع هریک از جهت‌ها را بازگرداند، روح به آن‌سمت حرکت می‌کند و اگر مقدارِ DIR_NONE را بازگرداند، روح از حرکت می‌ایستد.

[۳] نوعِ دادهٔ bool یک نوعِ داده‌ایست که به‌طورِ پیش‌فرض در c وجود ندارد و با استفاده از کتاب‌خانهٔ stdbool.h اضافه می‌شود. این نوع‌داده، دقیقاً همان int است، اما نامِ آن را تغییر داده‌ایم و قصد داریم تنها در آن مقادیرِ منطقیِ یک و صفر می‌ریزیم. همچنین در این‌کتاب‌خانه دو ثابت تعریف کرده‌ایم یکی true که برابرِ یک است و false که برابرِ صفر است.

همچنین توجه کنید که بررسیِ این‌که در مجاورتِ روح دیواری وجود دارد یا نه، با شماست اگر جهتی که تابع بازمی‌گرداند دیوار وجود داشته‌باشد، روح از دیوار عبور خواهدکرد که نباید این اتفاق رخ دهد.

۲.۴ decidePacman (۲۱ امتیاز)

این تابع نیز در physics.c وجود داردو به شکلِ زیر تعریف شده‌است.

```
Direction decidePacman(const Map* map, Pacman* pacman, Action action)
```

این تابع از هرجهت مشابهِ تابعِ decideGhost است، تنها فرقی که وجود دارد این است که حرکتِ پک‌من از کی‌برد ورودی داده می‌شود. به همین‌منظور این تابع، ورودیِ Action action را دارد که دکمهٔ فشرده‌شده برروی کی‌برد را نشان می‌دهد.
نوع‌دادهٔ Action در input.h به شکلِ زیر تعریف شده‌است.

```
#define MOVE_ACTION_MASK 0b001110

typedef enum {
    ACTION_LEFT = 0b001110,
    ACTION_RIGHT = 0b000110,
    ACTION_UP = 0b000010,
    ACTION_DOWN = 0b001010,

    // you have nothing to do with other actions, simply ignore them.
    ACTION_PAUSE = 0b010000,
    ACTION_IDLE = 0b000001,
    ACTION_EXIT = 0b100000
} Action;
```

نمایشِ 0bXXXXXXX عدد را به شکلِ دودویی بیان می‌کند و با نوشتنِ خودِ عدد هیچ‌تفاوتی ندارد.
ورودیِ تابع ممکن است هریک از Actionها باشد ولی درصورتی‌که یکی از ۴ جهت نباشد، شما باید از آن صرف‌نظر کنید و جهتِ حالِ حاضرِ پک‌من را بازگردانید.
(می‌توانید برای بررسیِ این‌که Action یکی از جهت‌های اصلی‌ست، از این واقعیت استفاده کنید که action & MOVE_MASK_ACTION عددی نابرابر با صفر خواهد شد.)
همچنین در این‌قسمت نیز باید به مواجهه با دیوارهای درونِ نقشه توجه شود. همان‌طور که احتمالاً در بازیِ پک‌من مشاهده‌کرده‌اید، درصورتی که پک‌من به سویی درحالِ حرکت باشد و دکمه‌ای روی کی‌برد فشرده‌شود، پک‌من تنها درصورتِ نبودِ دیوار تغییرِ جهت می‌دهد. همچنین اگر پک‌من در راستای حرکتِ خود به دیواری برخورد کند از حرکت می‌ایستد.

۵ منطق بازی

تمامی توابع این بخش در فایل game.c قرار می‌گیرند.

۱.۵ checkEatables (۱۹ امتیاز)

این تابع که به شکلِ زیر تعریف شد، خورده‌شدن خوراکی‌های مختلف (پنیر، گیلاس و آناناس) را بررسی می‌کند

```
void checkEatables(Map* map, Game* outGame, Pacman* outPacman, Ghost* outGhosts)
```

این تابع در هر cycle فراخوانی می‌شود، پس توجه کنید که در هنگامِ فراخوانیِ این‌تابع مکانِ پک‌من عددی اعشاری‌ست و دقیقاً در یک‌خانه نیست، پس این تابع باید طوری پیاده‌سازی شود که اگر فاصلهٔ پک‌من تا یک خوراکی کم‌تر از مقدارِ مشخصی بود، خوراکی خورده‌شود.
با خورده‌شدنِ خوراکی، باید از روی نقشه پاک شود و تعدادِ آن در بازی یکی کم‌تر شود همچنین اگر خوردنِ چیزی اثرِ دیگری نیز دارد، باید در این تابع اثراتِ خوردنِ آن را نیز اعمال شود.
تأثیراتِ خوراکی‌ها به شرحِ زیر است

۲.۵ checkGhostState (۶ امتیاز)

کارکردِ این تابع، چک کردن و درصورتِ لزوم تغییر دادنِ وضعیتِ روح از تدافعی به تهاجمی (از آبی به غیرِ آبی) است.
این تابع که به شکلِ زیر تعریف می‌شود **تنها یک روح** را ورودی می‌گیرد و وضعیتِ آن را بررسی و درصورتِ لزوم تغییر می‌دهد.

```
void checkGhostState(Ghost* ghost)
```

نامِ خوراکی	تأثیر
`cheese`	با خوردنِ پنیر، مقدارِ امتیاز به اندازهٔ `CHEESE_SCORE` افزایش می‌یابد.
`cherry`	با خوردنِ گیلاس، مقدارِ امتیاز به اندازهٔ `CHERRY_SCORE` افزایش می‌یابد.
`pineapple`	با خوردنِ آناناس، مقدارِ امتیاز به اندازهٔ `PINEAPPLE_SCORE` افزایش می‌یابد. همچنین همهٔ روح‌ها به حالتِ آبی می‌روند و به مدتِ `BLUE_DURATION` در این حالت می‌مانند. اگر روحی پیش از خوردنِ آناناس در حالتِ آبی بوده، در همین حالت می‌ماند و زمانِ باقی‌مانده‌ش دوباره برابرِ `BLUE_DURATION` خواهد شد.

در هر چرخهٔ بازی (`cycle`) این تابع به‌ازای هرروح یک‌بار (روی هم ۴ بار) فراخوانی می‌شود و در آن باید درصورتِ آبی بودنِ روح، مقدارِ `blueCounterDown` کاهش پیدا کند و درصورتِ تمام‌شدنِ زمانِ آبی بودنِ یک‌روح، باید آن راه به حالتِ تهاجمی (غیرِ آبی) تغییر دهد.
در صورتی که روحی در حالتِ تدافعی نباشد، این‌تابع نباید کاری انجام بدهد.

۳.۵ checkGhostCollision (۱۶ امتیاز)

این تابع به شکلِ زیر تعریف شده‌است.

```
void checkGhostCollision(Pacman* outPacman, Ghost* outGhost)
```

ورودی‌های آن، پک‌من و **تنها یک روح** خواهدبود. پس این تابع به ازای تک‌تکِ روح‌ها فراخوانی می‌شود و وظیفهٔ آن، بررسی برخورد پکمن با روح است.
در صورتی که برخورد رخ داده باشد باید با توجه به آبی بودن یا نبودن روح،پکمن و یا روح کشته شود و تغییرات مکانی صورت بگیرد.
به‌طورِ دقیق‌تر اگر روح کشته شود باید درخانه شروع و به صورت عادی(غیر آبی) ظاهر شود. اگر پکمن کشته شود باید به خانه شروع برود و از تعداد جان‌ها یکی کم شود.
برای بررسیِ برخورد باید به این‌نکته توجه کرد که مختصاتِ هردو (پک‌من و روح) اعشاری هستند و ممکن است که هرگز با هم برابر نشوند و باید با نزدیک‌شدنِ آن‌ها و کم‌تر شدنِ فاصله از حدی، برخورد را صورت دهیم. [۴]

۴.۵ isGameFinished (۳ امتیاز)

وظیفه این تابع بررسی اتمام بازی است. در صورتی که هیچ پنیر و هیچ آناناسی در صفحه باقی نمانده باشد (باقی‌ماندنِ گیلاس جلوی تمام‌شدنِ بازی را نمی‌گیرد) و یا درصورتی‌که جان‌های پک‌من تمام شده باشد بازی تمام می‌شود.
به محضِ این‌که این تابع، خروجی برابر با `true` یا همان یک را برگرداند بازی به پایان می‌رسد.

```
bool isGameFinished(Game* game, Pacman* pacman)
```

بارم‌بندی

این فاز در مجموع ۱۳۰ امتیاز است. ۹۸ امتیازِ آن مربوط به توابع است. این نمره به شرطِ کار کردنِ هر بخش داده می‌شود و در صورتِ کار نکردن ولی نوشته‌شدنِ کد، ٪۶۰ امتیاز تعلق می‌گیرد.
۲۲ امتیاز مربوط به تمیزیِ کد و ۱۰ امتیاز برای استفادهٔ صحیح از گیت درنظر گرفته‌شده‌است.
نکته: توجه کنید که همهٔ ۱۳۰ امتیازِ پروژه اجباری است و چیزی در این فاز امتیازی نیست

به سانِ رود
که در نشیبِ دره سر به سنگ می‌زند
رونده باش.
امیدِ هیچ معجزی ز مرده نیست،
زنده باش
ه.الف سایه

[۴] شما می‌توانید شرطِ برخورد را به شکل‌های متفاوتی پیاده‌سازی کنید. همچنین فاصلهٔ برخورد را نیز خودتان مشخص کنید.