دانشگاه هرمزگان

پروژه مبانی کامپیوتر و برنامه‌سازی

پیاده سازی بازی 2048

نیم‌سال دوم 1403

پروژه‌ی برنامه‌نویسی مقدماتی که پیش روی شماست، پیاده‌سازی منطق بازی محبوب 2048 است کد اصلی و رابط گرافیکی بازی به طور کامل برای شما فراهم شده و نیازی به تغییر در آن‌ها نیست (در صورت نیاز به هرگونه تغییر در ساختار پیاده شده ی بازی حتما با تیم حل تمرین مشورت کنید) وظیفه‌ی شما تکمیل توابع کلیدی است که در ادامه به سه فاز مجزا تقسیم شده‌اند. کد اصلی بازی در فایل `main.c` پیوست شده است.

آشنایی با بازی 2048

بازی 2048 در یک صفحه‌ی شبکه‌ای 4در4 انجام می‌شود که از کاشی‌های شماره‌دار پر شده است. هدف شما این است که با ترکیب این کاشی‌ها، به کاشی با عدد 2048 برسید، هرچند پس از رسیدن به آن نیز می‌توانید برای کسب امتیاز بالاتر به بازی ادامه دهید.

نحوه‌ی انجام بازی

• برای حرکت دادن کاشی‌ها، از کلیدهای جهت‌نما (بالا، پایین، چپ، راست) روی صفحه‌کلید خود استفاده می‌کنید.

• کاشی‌هایی که شماره یکسان دارند و هنگام حرکت با یکدیگر برخورد می‌کنند، با هم ادغام شده و به یک کاشی واحد با مجموع مقادیرشان تبدیل می‌شوند (مثلاً 2+2=4، 4+4=8).

• پس از هر حرکت، یک کاشی جدید با مقدار 2 یا 4 به صورت تصادفی در یک خانه‌ی خالی از صفحه ظاهر می‌شود.

• بازی زمانی به پایان می‌رسد که دیگر هیچ جای خالی در صفحه نباشد و هیچ ادغام دیگری ممکن نباشد. اگر موفق به ساخت کاشی 2048 شوید، برنده هستید، در غیر این صورت بازنده‌ی بازی خواهید بود.

قوانین بازی

• **صفحه بازی**: بازی روی یک شبکه‌ی 4در4 انجام می‌شود که هر خانه آن حاوی یک کاشی عددی است. این اعداد از عدد 2 شروع می‌شوند و توان‌هایی از 2 هستند.

• **کاشی‌های آغازین**: در شروع بازی، دو کاشی با اعداد 2 یا 4 به صورت تصادفی در صفحه قرار می‌گیرند.

• **حرکت کاشی‌ها**: شما می‌توانید کاشی‌ها را در چهار جهت بالا، پایین، چپ و راست حرکت دهید. تمام کاشی‌های روی صفحه در جهت انتخابی شما حرکت می‌کنند تا به لبه‌ی صفحه یا کاشی دیگری برخورد کنند.

• **ترکیب کاشی‌ها**: زمانی که دو کاشی با عدد یکسان در جهت انتخابی شما به هم برخورد می‌کنند، آن‌ها به یک کاشی واحد با مجموع مقادیرشان تبدیل می‌شوند. برای مثال، اگر دو کاشی با عدد 2 با هم برخورد کنند، به یک کاشی 4 تبدیل می‌شوند.

• **افزودن کاشی‌های جدید**: بعد از هر حرکت، یک کاشی جدید با مقدار 2 یا 4 به صورت تصادفی در یک خانه‌ی خالی به صفحه اضافه می‌شود.

• **هدف بازی**: هدف اصلی، ترکیب کاشی‌ها برای رسیدن به کاشی 2048 است. با این حال، بازی می‌تواند فراتر از 2048 نیز ادامه پیدا کند و هدف نهایی کسب بالاترین امتیاز ممکن است.

برای اطلاعات بیشتر می‌توانید به صفحه‌ی ویکی‌پدیا 2048 مراجعه کنید

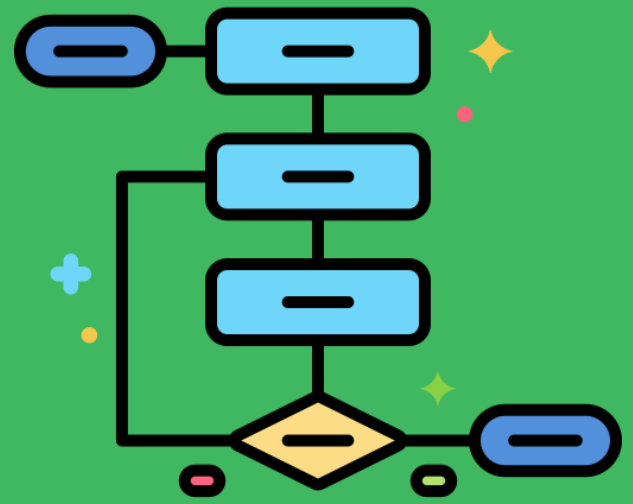
https://shorturl.at/qowgq

می‌توانید نسخه‌ی آنلاین بازی را در لینک زیر تجربه کنید

https://play2048.co/classic

فاز اول

پیاده‌سازی منطق اصلی بازی



در این فاز، شما هسته اصلی بازی 2048 را خواهید ساخت. این شامل مدیریت حافظه برای صفحه‌ی بازی، اضافه کردن کاشی‌های تصادفی، مقداردهی اولیه بازی، انجام حرکات و بررسی پایان بازی می‌شود.

AllocateBoard

هدف: این تابع مسئول تخصیص حافظه برای صفحه‌ی بازی است. صفحه‌ی بازی به صورت یک ماتریس (آرایه دو بعدی) نمایش داده می‌شود.

ورودی: `int size` - یک عدد صحیح که ابعاد (طول و عرض) صفحه‌ی بازی مربع را مشخص می‌کند.

خروجی مورد انتظار: یک اشاره‌گر به اشاره‌گر اعداد صحیح `(int**)` که همان صفحه‌ی بازی تخصیص داده شده است.

نکات و توضیحات: در این فاز هدف شما فقط تخصیص حافظه و پیاده سازی منطق بازی برای یک صفحه 4*4 است.

FreeBoard

هدف: این تابع مسئول آزادسازی حافظه‌ای است که قبلاً برای صفحه‌ی بازی توسط AllocateBoard تخصیص داده شده بود.

ورودی:

int **board - اشاره‌گر به آرایه دوبعدی (ماتریس) از اعداد صحیح که قبلاً توسط تابع AllocateBoard ساخته شده.

int size- ابعاد صفحه‌ی بازی.

خروجی مورد انتظار: هیچ (void).

نکات و توضیحات:

هدف اصلی این تابع، جلوگیری از نشت حافظه (Memory Leak) با بازگرداندن حافظه به سیستم است.

AddRandomTile

هدف: این تابع یک کاشی جدید (با مقدار 2 یا 4) را به صورت تصادفی در یکی از خانه‌های خالی صفحه‌ی بازی اضافه می‌کند.

ورودی: `GameState * state` - یک اشاره‌گر به ساختار `GameState` که وضعیت فعلی بازی شامل صفحه، امتیاز و ... را در بر می‌گیرد.

خروجی مورد انتظار: هیچ `(void)`.

نکات و توضیحات:

اگر هیچ خانه‌ی خالی در صفحه وجود نداشت، این تابع نباید عملی انجام دهد.

در بازی 2048، احتمال ظاهر شدن عدد 2، 90 درصد و عدد 4، 10 درصد است.

InitGame

هدف: این تابع بازی را برای شروع آماده و مقداردهی اولیه می‌کند.

ورودی:

`GameState * state` - یک اشاره‌گر به ساختار `GameState` که باید مقداردهی اولیه شود.

`int size` - ابعاد مورد نظر صفحه‌ی بازی.

خروجی مورد انتظار: هیچ `(void)`.

نکات و توضیحات:

وضعیت اولیه‌ی امتیاز و وضعیت پایان بازی در این تابع تعیین می‌شود.

شروع بازی 2048 همیشه با دو کاشی اولیه است.

Move

هدف: این تابع منطق اصلی حرکت و ادغام کاشی‌ها در صفحه را بر عهده دارد.

ورودی:

`GameState * state` - اشاره‌گر به ساختار `GameState` که صفحه‌ی بازی و امتیاز در آن تغییر خواهند کرد.

`int dir` - یک عدد صحیح که جهت حرکت را مشخص می‌کند: `0` برای بالا، `1` برای راست، `2` برای پایین، `3` برای چپ.

خروجی مورد انتظار: یک مقدار بولی `(bool)` که اگر حرکت منجر به تغییری در صفحه (جابجایی یا ادغام کاشی‌ها) شد، `true` و در غیر این صورت `false` را برمی‌گرداند.

نکات و توضیحات:

این تابع باید جابجایی و ادغام کاشی‌ها را در یک جهت معین مدیریت کند.

پس از انجام عملیات، امتیاز بازی باید به درستی به‌روزرسانی شود.

یک کاشی تنها یک بار در هر حرکت می‌تواند ادغام شود.

IsGameOver

هدف: این تابع بررسی می‌کند که آیا بازی به پایان رسیده است یا خیر (یعنی هیچ حرکت ممکن دیگری وجود ندارد).

ورودی: `GameState * state` - اشاره‌گر به ساختار `GameState` .

خروجی مورد انتظار: یک مقدار بولی `(bool)` که اگر بازی تمام شده باشد، `true` و در غیر این صورت `false` را برمی‌گرداند.

نکات و توضیحات:

بازی زمانی تمام می‌شود که هیچ خانه‌ی خالی برای اضافه کردن کاشی جدید وجود نداشته باشد و هیچ کاشی مجاوری هم برای ادغام موجود نباشد.

CopyBoard

تابع `void CopyBoard(int **dst, int **src, int size)` در این پروژه ارائه شده است. این تابع می‌تواند برای کپی کردن محتویات یک ماتریس به ماتریس دیگر استفاده شود و در پیاده‌سازی توابع دو فاز بعدی مفید خواهد بود اما الزامی نیست و صرفا در صورت نیاز با هماهنگی قابل استفاده است.

فاز دوم

مدیریت تاریخچه (Undo/Redo)

در این فاز، شما قابلیت‌های بازگشت به عقب (Undo) و تکرار به جلو (Redo) را به بازی اضافه خواهید کرد. این ویژگی به کاربر امکان می‌دهد تا حرکات قبلی خود را لغو کرده یا دوباره اعمال کند.

SaveState

هدف: این تابع وضعیت فعلی بازی را در ساختار تاریخچه (History) ذخیره می‌کند تا امکان بازگشت به آن فراهم شود.

ورودی:

History *h - اشاره‌گر به ساختار History که آرایه‌ای از GameState ها را برای تاریخچه ذخیره می‌کند.

GameState *s - اشاره‌گر به GameState فعلی که باید ذخیره شود.

خروجی مورد انتظار: هیچ (void).

نکات و توضیحات:

این تابع باید تضمین کند که تاریخچه‌ی بازی به درستی مدیریت می‌شود، از جمله حفظ ترتیب حالت‌ها و مدیریت وضعیت‌هایی که کاربر پس از Undo یک حرکت جدید انجام می‌دهد.

Undo

هدف: این تابع بازی را به حالت ذخیره شده‌ی قبلی بازمی‌گرداند.

ورودی:

`GameState *s` - اشاره‌گر به `GameState` فعلی که باید به روز شود.

`History *h` - اشاره‌گر به ساختار `(History)`.

خروجی مورد انتظار: هیچ `(void)`.

نکات و توضیحات:

اگر هیچ حالت قبلی برای بازگشت وجود نداشته باشد، این تابع نباید هیچ عملی انجام دهد.

پس از بازگشت، وضعیت بازی (صفحه، امتیاز، وضعیت پایان بازی) باید دقیقاً مطابق با حالت ذخیره شده قبلی باشد.

Redo

هدف: این تابع بازی را به حالت ذخیره شده‌ی بعدی (اگر پس از Undo وجود داشته باشد) پیش می‌برد.

ورودی:

GameState *s - اشاره‌گر به GameState فعلی که باید به روز شود.

History *h - اشاره‌گر به ساختار (History).

خروجی مورد انتظار: هیچ (void).

نکات و توضیحات:

اگر هیچ حالت آینده‌ای برای پیشروی وجود نداشته باشد، این تابع نباید هیچ عملی انجام دهد.

مانند Undo، وضعیت بازی باید پس از Redo به درستی به‌روزرسانی شود.

فاز سوم

تغییر اندازه‌ی صفحه‌ی بازی (نمره‌ی اضافی)

این فاز اختیاری است و برای نمره‌ی اضافی در نظر گرفته شده است در نتیجه صرفا در صورت هماهنگی با گروه حل تمرین قابل اجرا می باشد. در این بخش، شما باید قابلیت تغییر اندازه‌ی صفحه‌ی بازی را فراهم کنید.

boardSize

هدف: این متغیر ابعاد صفحه‌ی بازی را تعیین می‌کند.

ورودی: این یک متغیر int است که به طور پیش‌فرض 4 است.

خروجی مورد انتظار: ندارد (این یک متغیر است).

توضیحات:

برای دریافت نمره‌ی اضافی این فاز، شما باید مکانیزمی را پیاده‌سازی کنید تا مقدار boardSize قابل تغییر باشد (مثلاً از طریق ورودی کاربر).

پیاده‌سازی شما باید به گونه‌ای باشد که بازی با هر اندازه‌ی صحیحی که به boardSize اختصاص داده می‌شود، به درستی کار کند. این شامل تمام عملیات‌های مربوط به صفحه (تخصیص، آزادسازی، حرکت، بررسی پایان بازی، و مدیریت تاریخچه) می‌شود.

هر گونه سوال یا ابهام را از طریق صفحه کوئرا درس با گروه حل تمرین در میان بگذارید.

با آرزوی موفقیت

تیم حل تمرین نیم‌سال دوم ۱۴۰۳