| پروژه  **شطنرد** |
| --- |

| ارائه دهنده | عرفان اعتصامی |
| --- |

* [ا) الگوریتم کلی برنامه](#الگوریتم)
* [۲) ثوابت](#ثوابت)
  + تعریف و کاربرد
* [۳) متغیرهای سرتاسری](#سرتاسری)
  + تعریف و کاربرد
* [۴) توابع](#loadBoard)
  + [تعریف و کاربرد تابع loadBord](#loadBoard)
  + [تعریف و کاربرد تابع printBoard](#printBoard)
  + [تعریف و کاربرد تابع findFreePiece](#findFreePiece)
  + [تعریف و کاربرد تابع createHolesAndCoins](#createHolesAndCoins)
  + [تعریف و کاربرد تابع checkForCoinsRelease](#checkForCoinRelease)
  + [تعریف و کاربرد تابع freeThePiece](#freeThePiece)
  + [تعریف و کاربرد تابع obtainMove](#obtainMove)
  + [تعریف و کاربرد تابع selectDice](#selectDice)
  + [تعریف و کاربرد تابع anyMoveLeft](#anyMoveLeft)
  + [تعریف و کاربرد تابع movePlayer](#movePlayer)
  + [تعریف و کاربرد تابع isPlayerCastleFull](#isPlayerCastleFull)
  + [تعریف و کاربرد تابع movePieceToO](#movePieceToO)
  + [تعریف و کاربرد تابع showMsg](#showMsg)
  + [تعریف و کاربرد تابع loadNames](#loadNames)
  + [تعریف و کاربرد تابع save](#save)
  + [تعریف و کاربرد تابع main](#main)

در ابتدا، صفحه‌ی بازی را به ۲ قسمت کلی تقسیم می‌نماییم: ۱)قلمرو بازیکن۱(C1 و R1) و ۲)قلمرو بازیکن۲(C2 و R2) که هر کدام از این بخش ها شامل ۱۲ ستون(هر قلعه ۶ ستون و هر جاده نیز ۶ ستون) هستند. ظرفیت هر ستون را ۶ در نظر گرفته ایم؛ یعنی، در هر ستون حداکثر می‌توان ۶ مهره را قرار داد. به این گونه صفحه‌ی بازی را می‌توان به صورت یک آرایه‌ی سه بعدی در نظر گرفت.

هر بازیکن ۱۵ مهره از ۶ نوع سرباز، فیل، اسب، قلعه، ملکه و شاه دارد. به هر نوع مهره از هر بازیکن، عددی از ۱ تا ۱۲ را نسبت می‌دهیم و خانه‌های آرایه‌ی سه بعدی صفحه‌ی بازی را طبق نقشه‌ی اولیه، مقداردهی می‌کنیم. حال می‌توانیم با انجام عملیات ریاضی مختلف و جابه‌جایی اعداد با یکدیگر، بازی و قوانین آن را پیاده سازی کنیم.

حرکت هر بازیکن در قلمرو خودش به سمت چپ و در قلمرو حریف به سمت راست می‌باشد.

ثابت EMPTY :

مقدار صفر۰ دارد و بیانگر این است که آرایه مورد نظر از خانه‌ها خالی از مهره است. به عنوان مثال آرایه‌ی سه بعدی صفحه‌ی بازی یا آرایه‌ی نشان دهنده‌ی مهره‌های زندانی در ابتدا خالی هستند.

ثوابت مهره ها :

به مهره‌های بازیکن۱، اعداد ۱ تا ۶ را به ترتیب به سرباز، فیل، اسب، قلعه، ملکه و شاه نسبت می‌دهیم و اعداد ۷ تا ۱۲ را به همین ترتیب به مهره‌های بازیکن۲ نسبت می‌دهیم؛ به این ترتیب می‌توانیم با مهره ها مانند اعداد برخورد کنیم.

ثابت ROWS : مقدار ۲ دارد. صفحه‌ی بازی شامل ۲ قسمت کلی است: قلمرو بازیکن۱ و قلمرو بازیکن ۲.

ثابت COLUMNS : مقدار ۱۲ دارد. هر قسمت شامل ۱۲ ستون می‌باشد.

ثابت CAPACITY : مقدار ۶ دارد. هر ستون حداکثر گنجایش ۶ مهره را دارد.

ثابت PIECES : مقدار ۱۵ دارد. هر بازیکن ۱۵ مهره در اختیار دارد.

ثابت TYPES : مقدار ۶ دارد. این ۱۵ مهره شامل ۶ نوع سرباز، فیل، اسب، قلعه، ملکه و شاه هستند.

ثوابت داده‌ها و نتایج درست یا نادرست :

برای ساده‌تر شدن اجرای قوانین بازی، برای حالات و حرکات نادرست، ثوابتی که مقادیر پرتی نسبت به مقادیر ثوابت مربوط به اجزاء اصلی بازی دارند در نظر گرفته ایم. برای حرکت نادرست(WRONG\_MOVE) که مقدار ۱۰۰- دارد. برای حالتی که حرکتی برای بازیکن باقی نمانده(NO\_MOVES) که مقدار ۲۰۰- دارد. برای تاسی که استفاده شده(USED) که مقدار ۳۰۰- دارد. برای حالت زندانی بودن مهره(OUT) که مقدار ۱- دارد. برای حالتی که تابع وظیفه اش را به درستی انجام داده است(DONE) که مقدار ۱۰۰ دارد.

ثوابت مربوط به ذخیره‌ی بازی :

برای انجام درست حالاتی که کاربر می‌خواهد بازی را ذخیره کند و شماره گذاری فایل‌های دخیره شده، نیز ثوابتی را در نظر گرفته ایم. ثابت های SHOW و HIDE که به ترتیب مقادیر ۱۰۰ و ۱۰۰- را دارند. برای حالتی که ممکن است در روند ذخیره سازی مشکلی پیش آید(SAVE\_INTERRUPTED) که مقدار ۱۰۰۰- دارد. همچنین برای نام و شماره گذاری فایل ذخیره شده، ثوابت SavedGame و SavedGameNumber را تعریف کرده ایم که اطلاعات را در قالب فرمت .dat ذخیره می‌کنند.

متغیر نام مهره ها(char \*names) :

این متغیر که آرایه ای از اشاره‌گرها می‌باشد شامل ۱۳ عنصر است که هر عنصر آن اشاره‌گری به نام مهره ها است که دارای ترتیبی مشابه ثوابت مهره‌ها می‌باشد. به جهت هماهنگ کردن شماره‌ی این عناصر با ثوابت مربوط به آن‌ها یعنی از ۱ تا ۱۲؛ با توجه به این که اندیس یک آرایه از صفر۰ آغاز می‌شود، عنصر نخست این آرایه را به صورت یک رشته که دارای ۴ فاصله(space) می‌باشد تعریف کرده‌ایم.

متغیر نام خانه ها(char \*rooms) :

این متغیر نیز آرایه‌ای از اشاره‌گرها است که شامل ۲۵ عنصر می‌باشد که هر عنصر آن اشاره‌گری به نام ستون های هر قلمرو از صفحه‌ی بازی است(C1\_1 تا C1\_6، R1\_1 تا R1\_6، C2\_1 تا C2\_6 و R2\_1 تا R2\_6) و عنصر آخر آن O می‌باشد که مربوط به بیرون بردن مهره‌ها از بازی برای پیروزی است.

متغیر نام مهره‌های زندانی(char \*oNames) :

این متغیر نیز آرایه‌ای از اشاره‌گرها است که شامل ۱۳ عنصر است. هر عنصر آن یک اشاره‌گر به نوع مهره‌ی زندانی شده‌ی هر بازیکن است(O\_S تا O\_K برای بازیکن۱ و O\_S تا O\_K برای بازیکن۲). جهت هماهنگی شماره‌ی این مهره های زندانی شده با ثوابت مهره‌ها یعنی از ۱ تا ۶ برای بازیکن۱ و از ۷ تا ۱۲ برای بازیکن۲؛ با توجه به این اندیس های آرایه از صفر۰ شروع می‌شوند، عنصر نخست این آرایه را به صورت یک رشته که دارای ۳ فاصله(space) می‌باشد تعریف کرده ایم.

آرایه‌ی سه بعدی صفحه‌ی بازی(int board) :

بعد اول این آرایه با توجه به ثابت ROWS مقادیر صفر۰(قلمرو بازیکن ۱) و ۱(قلمرو بازیکن۲) را اختیار می‌کند، بعد دوم با توجه به ثابت COLUMNS مقادیر صفر۰ تا ۱۱، بعد سوم با توجه به ثابت CAPACITY مقادیر صفر۰ تا ۵. این آرایه در ابتدا خالی است و سپس خانه‌های آن با توجه به ورودی های نخستین بازی و جابه‌جایی مهره‌ها، مقادیر عددی ۱ تا ۱۲ را با توجه به ثوابت مهره‌ها به خود می‌گیرند و خانه های خالی آن با مقدار صفر۰ پر می‌شوند.

آرایه‌ی دو بعدی مهره‌های زندانی(int playerOs) :

بعد نخست این آرایه با توجه به این که دو بازیکن داریم مقادیر صفر۰(بازیکن ۱) و ۱(بازیکن ۲) را اختیار می‌کند، بعد دوم با توجه به ثابت PIECES مقادیر صفر۰ تا ۱۴(۱۵ مهره). در ابتدا، این آرایه نیز خالی است اما در صورت زندانی شدن مهره‌ای از هر بازیکن، خانه های آن پر می‌شوند.

آرایه‌ی های دو بعدی چاله و سکه(int holes, int coins) :

این دو آرایه ساختاری مشابه با یکدیگر دارند. به این صورت که بعد اول آن ها ۳ مقدار (صفر۰ تا ۲) را به خود می‌گیرد زیرا با توجه به قوانین بازی سه خانه از صفحه‌ی بازی به صورت تصادفی دارای چاله یا سکه می‌شود. بعد دوم که برای آدرس دهی به‌کار می‌رود نیز دارای دو مقدار صفر۰ و ۱ می‌باشد که عنصر صفر۰ بیانگر قلمرویی است که چاله یا سکه در آن قرار دارد و عنصر ۱ بیانگر آن ستون.

آرایه‌ی یک بعدی سکه های جمع شده(int coinsCollected) :

این آرایه دو عنصری همانطور که از نام‌ آن پیدا است، تعداد سکه های جمع شده‌ی توسط هر بازیکن را در خود ذخیره می‌کند. عنصر صفر۰ برای شمارش سکه‌های بازیکن۱ و عنصر ۱ برای سکه‌های بازیکن۲ می‌باشد. در صورتی که بازیکن در خانه‌ای قرار گیرد که شامل سکه است، این سکه به تعداد سکه های آن بازیکن افزوده شده و سپس از صفحه‌ی بازی حذف می‌شود و در صورتی که مهره‌ای از این بازیکن زندانی شود، این بازیکن مختار است که از سکه‌هایش برای آزاد کردن آن مهره‌اش بهره ببرد.

آرایه‌ی یک بعدی امتیاز(int point) :

این آرایه دو عنصری در پایان بازی برای شمارش تعداد مهره‌هایی که هر بازیکن برای پیروزی به بیرون صفحه‌ی بازی منتقل می‌کند به کار می رود؛ به این صورت که، با خارج شدن هر مهره‌ از بازیکن، ۱ واحد به امتیاز آن بازیکن افزوده می‌شود(عنصر صفر۰ برای بازیکن۱ و عنصر۱ برای بازیکن۲). هنگامی که یک بازیکن تمام مهره‌های خودش را خارج کند، امتیاز وی ۱۵ شده و آن بازیکن پیروز می‌شود و بازی به پایان می‌رسد.

آرایه‌ی یک بعدی ممنوعیت حرکت(int playerBanned) :

این آرایه نیز دارای دو عنصر است(عنصر صفر۰ برای بازیکن۱ و عنصر ۱ برای بازیکن۲). در صورتی که مهره‌ی یک بازیکن در چاله بیفتد آن بازیکن برای دو نوبت اجازه‌ی حرکت ندارد. این آرایه برای کنترل ممنوعیت حرکت هر بازیکن به‌کار می‌رود.

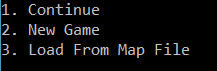
متغیر newSaveNumber :

این متغیر در ابتدا مقدار ۱ را دارد و برای شماره‌گذاری فایل های ذخیره شده به ترتیب به‌کار می‌رود.

وظیفه : بارگذاری بازی به سه روش: ۱)فایل‌های ذخیره‌شده ۲)بازی جدید ۳)استفاده از نقشه.

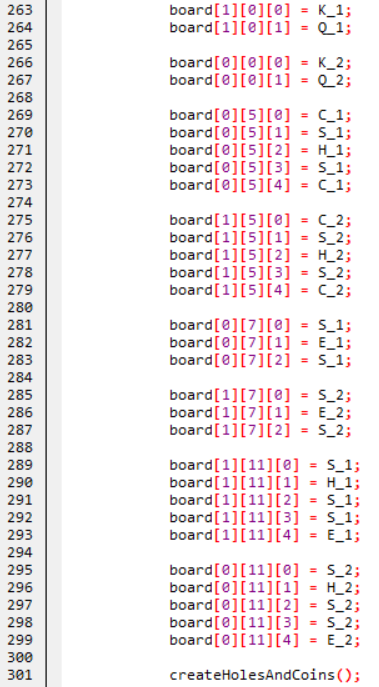
ورودی : ندارد.

خروجی : متغیر صحیح turn که بیانگرنوبت است.



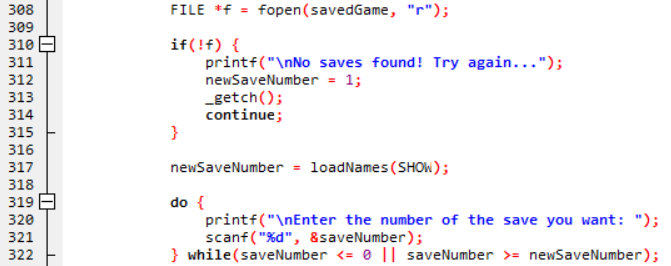
در ابتدا این تابع باعث چاپ یک خروجی به شکل بالا می‌شود که از کاربرد می‌خواهد یکی از ۳ روش گفته شده را انتخاب کند.

حالت بازی جدید :

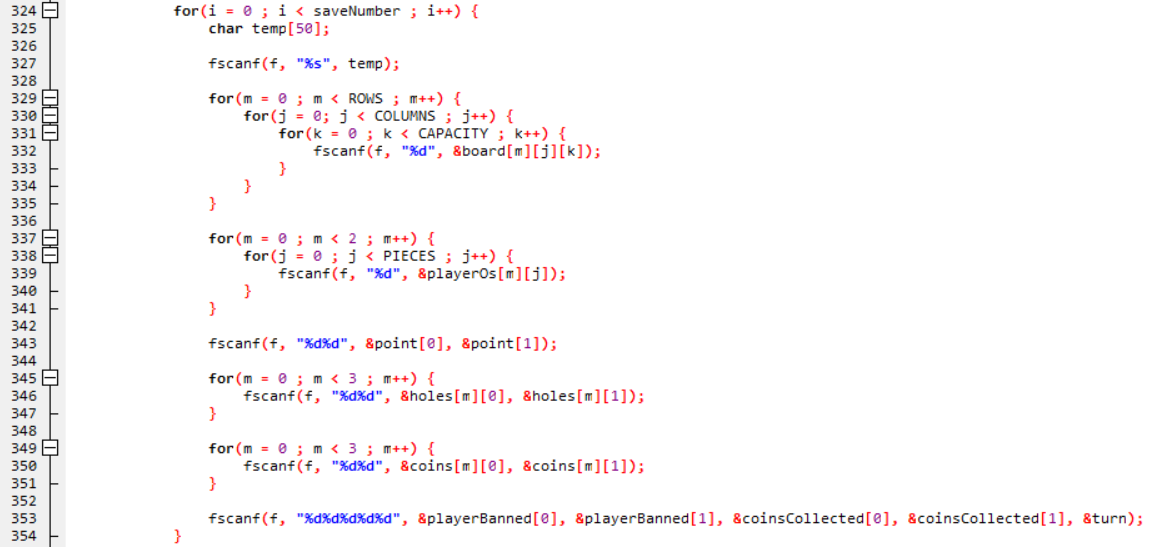


خانه‌های آرایه‌ی سه بعدی صفحه‌ی بازی که شامل مهره هستند، طبق نقشه‌ی اولیه‌ی بازی و مطابق با ثوابت مهره‌ها مقداردهی می‌شوند و سپس چاله‌ها و سکه‌ها در مکان‌های تصادفی از صفحه‌ی بازی قرار می‌گیرند.

حالت فایل‌های ذخیره‌شده :

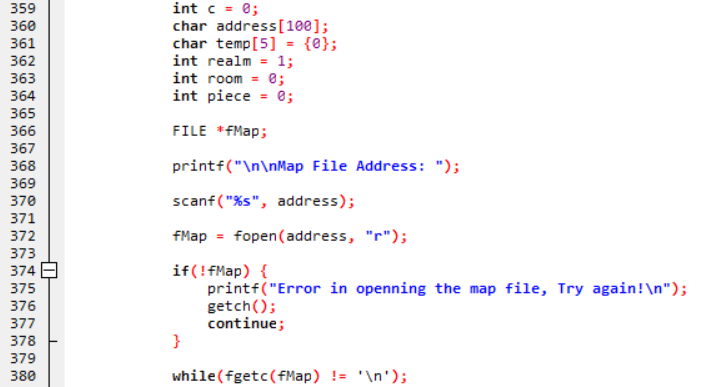


در ابتدا فایل ذخیره‌ی بازی که ثابت savedGame به آن اشاره می‌کند برای خواندن(پرچم r) باز می‌شود و در صورت عدم وجود فایل ذخیره، پیام مناسب چاپ می‌شود و در صورت وجود چند فایل از کاربر خواسته می‌شود که شماره‌ی آن فایل را که در لیست نمایان‌شده مشخص است، انتخاب کند.



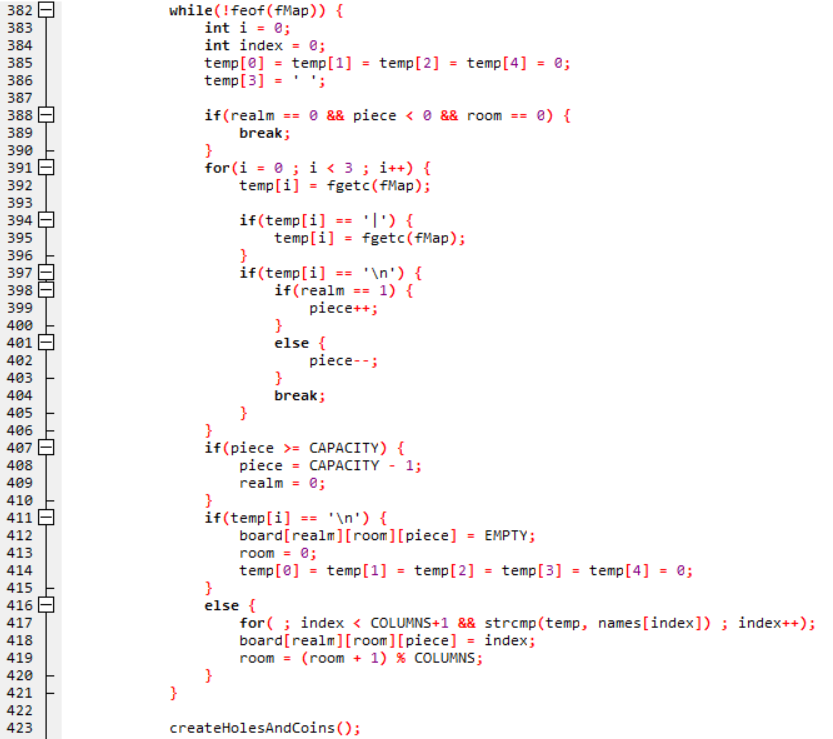
سپس صفحه‌ی بازی، مهره‌های زندانی، مهره‌های خارج شده، چاله‌ها، سکه‌ها، ممنوعیت حرکت بازیکن‌ها، سکه‌های جمع‌شده توسط هر بازیکن و نوبت بازی خوانده می‌شود.

حالت استفاده از نقشه :



در این حالت، از کاربر خواسته می‌شود که آدرس فایل نقشه را تایپ کند و در صورتی که فایل به درستی باز نشود یا فایل پیدا نشود، پیام مناسب در خروجی چاپ می‌شود.

خط ۳۸۰ : سطر اول فایل نقشه خوانده می‌شود.



سپس خطوط موجود در فایل نقشه، کاراکتر به کاراکتر تا هنگامی که به نمایه‌ی پایان فایل برسد، خوانده می‌شوند. سپس در پایان نیز مکان چاله‌ها و سکه‌ها به صورت تصادفی مشخص می‌شود.

خط ۳۹۱ : دلیل این که i < 3 قرارداده‌ایم، خواندن نام مهره‌ها است که ۳ کاراکتری هستند.

وظیفه : چاپ صفحه‌ی بازی به همراه جدول مهره‌های زندانی

ورودی : ندارد.

خروجی : ندارد.



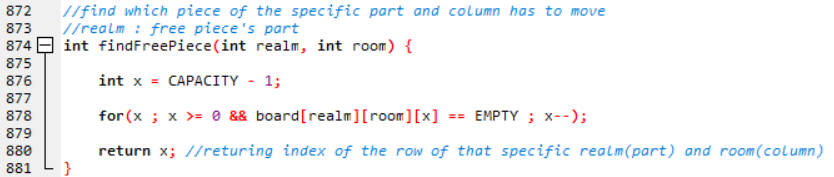
متغیر outCounter تعریف شده در ابتدای تابع برای چاپ کردن به ترتیب مهره‌های زندانی به صورت زیر هم به‌کار می‌رود. فرآیند چاپ صفحه‌ی بازی به دو جفت حلقه‌ی تودرتو تقسیم شده است که جفت اول، قسمت بالای صفحه را(قلمرو بازیکن۲) و جفت دوم قسمت پایین صفحه را(قلمرو بازیکن۱) چاپ می‌کند.

خط ۴۴۴ : برای جلوگیری از شلوغ شدن محیط اجرای بازی و عدم نیاز به اسکرول، هرگاه این تابع فراخوانی شود، در ابتدای کار محیط اجرای بازی توسط این دستور پاک می‌شود.

وظیفه : یافتن روترین مهره‌ی قابل حرکت

ورودی : متغیر صحیح realm(قلمرو) و متغیر صحیح room(ستون).

خروجی : متغیر صحیح x که بیانگر جای روترین مهره‌ی قابل حرکت در قلمرو و ستون مورد نظر است.

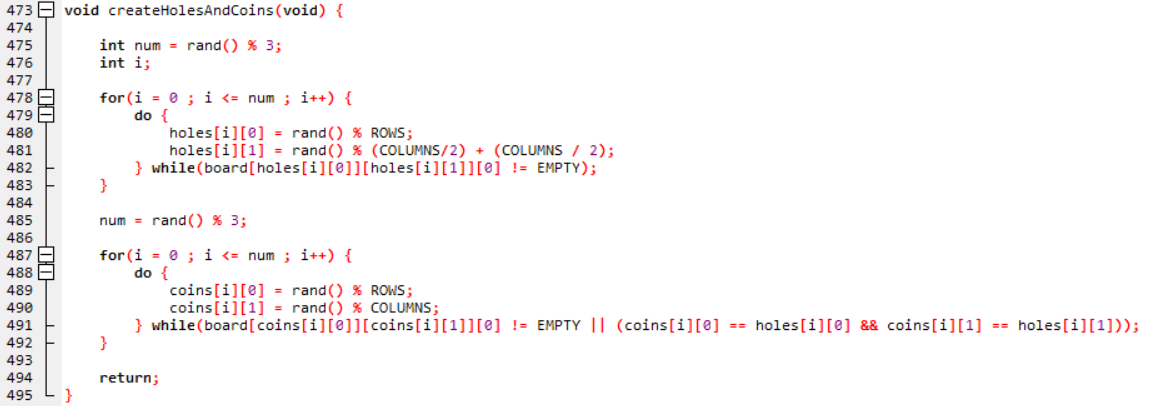


چون اندیس های آرایه از صفر۰ آغاز می‌شوند، در ابتدا x را برابر با CAPACITY-1 قرار دادیم. حلقه‌ی داخل تابع روترین مهره را مکان‌یابی کرده و باز‌می‌گرداند.

وظیفه : ایجاد چاله و سکه در مکان‌های تصادفی

ورودی : ندارد.

خروجی : ندارد.



در ابتدا عدد رندومی بین صفر۰ تا ۲ تولید شده و در متغیر num ذخیره می‌شود. سپس حلقه‌ی for اول در تابع اجرا شده که حلقه do\_while داخل آن وظیفه‌ی آدرسی دهی تصادفی به مکان چاله ها را دارد. سپس همین روند برای تولید سکه‌ها نیز به‌کار می‌رود.

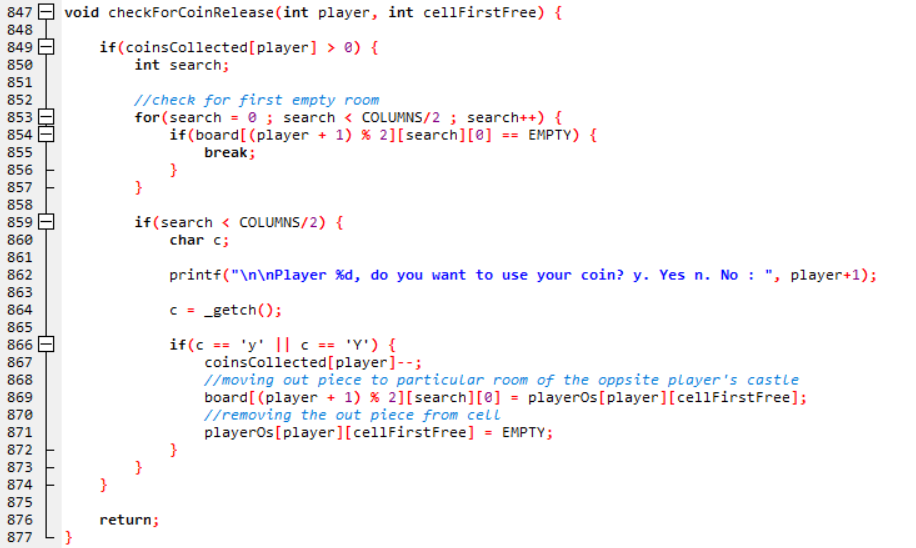
خط ۴۸۱ : دلیل افزودن COLUMNS/2 این است که در قوانین گفته شده که چاله‌ها در قسمت‌های جاده‌ی صفحه‌ی بازی قرار می‌گیرند.

خط ۴۹۱ : قسمت دوم شرط داخل پرانتز‌های while به دلیل هم‌نهشتی احتمالی جای چاله‌ها و سکه‌هاست.

وظیفه : آزادسازی اختیاری مهره‌ی زندانی شده و جاگذاری آن در یکی از خانه‌های قلعه‌ی بازیکن حریف.

ورودی : متغیر صحیح player(صفر۰ برای بازیکن۱ و ۱ برای بازیکن۲) و متغیر cellFirstFree که بیانگر روترین مهره‌ی زندانی شده در جدول مهره‌های زندانی است.

خروجی : ندارد.



در ابتدای کار بررسی می‌شود که آیا اصلاً بازیکن مشخص شده، سکه‌ای برای استفاده دارد یا خیر. سپس در صورت مثبت بودن جواب، حلقه موجود با استفاده از متغیر search به دنبال نخستین ستون خالی در قلعه بازیکن حریف می‌گردد. در صورت یافت شدن این ستون، از بازیکن پرسیده می‌شود که آیا قصد استفاده از سکه‌ی خود را دارد یا خیر. در صورت مثبت بودن پاسخ، اولین مهره‌ی موجود در زندان به خانه‌ی پیدا شده در صفحه‌ی بازی منتقل می‌شود.

وظیفه : آزادکردن مهره‌ها از زندان و قرار دادن آن‌ها در جای مناسب از خانه‌های قلعه‌ی حریف.

ورودی : متغیر صحیح player(صفر۰ برای بازیکن۱ و ۱ برای بازیکن۲)، متغیر صحیح pIndex که بیانگر اندیس مهره است و متغیر صحیح dice که بیانگر مقدار تاس است.

خروجی : مقدار WRONG\_MOVE برای نادرستی حرکت و مقدار DONE در صورت انجام شدن درست فرآیند.

فقط به شرح قسمت‌هایی که شاید کمی مبهم باشند، می‌پردازیم.

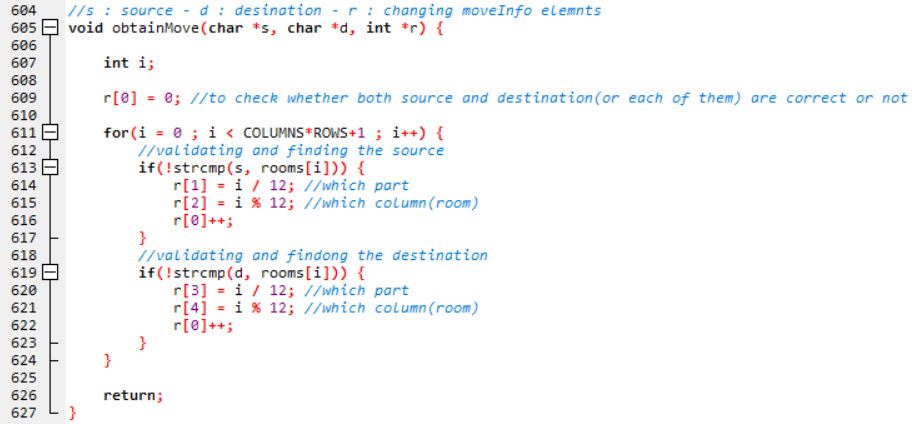


این حلقه برای این است که همواه مطمئن باشیم، اولین مهره‌ی زندانی در عنصر صفر۰ام زندان قرار دارد.

وظیفه : تشخیص مبدأ و مقصد حرکت.

ورودی : آرایه یک بعدی s که بیانگر آدرس مبدأ ورودی کاربر است، آرایه یک بعدی d که بیانگر آدرس مقصد ورودی کاربر است و آرایه یک بعدی r که آدرس مبدأ و مقصد را در خود ذخیره می‌کند.

خروجی : ندارد.

منظور از آدرس، قلمرو و ستون است.

حلقه‌ی for، وظیفه‌ی مطابقت دادن ستون مبدأ و مقصد وارده را با ستون‌های پیش‌فرض بازی دارد.

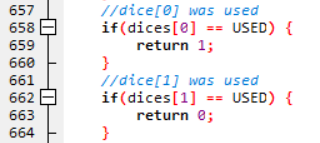
خط ۶۰۹ : عنصر صفر۰ام این آرایه برای مشخص کردن درستی یا نادرستی مبدأ و مقصد وارده به‌کار می‌رود به گونه‌ای که اگر هر دو درست باشند مقدار ۲، فقط یک‌کدام، مقدار ۱ و اگر هیچ‌کدام درست نباشند، مقدار صفر۰ را دارد.

وظیفه : جابه‌جایی پذیری تاس‌ها و مطابقت تاس‌ها با حرکت.

ورودی : متغیر صحیح player(صفر۰ برای بازیکن۱ و ۱ برای بازیکن۲)، متغیرهای صحیح sRealm، sRoom، dRealm، dRoom که به ترتیب نمایانگر قلمرو مبدأ، ستون مبدأ، قلمرو مقصد، ستون مقصد هستند، آرایه یک بعدی dices که حاوی مقدار تاس‌هاست و اشاره‌گر matched که قرار است آن را در صورت مطابقت یا عدم مطابقت بازگردانی کنیم.

خروجی : مقدار صفر۰ برای عنصر نخست آرایه‌ی تاس‌ها و مقدار ۱ برای عنصر دوم آرایه‌ی تاس‌ها.

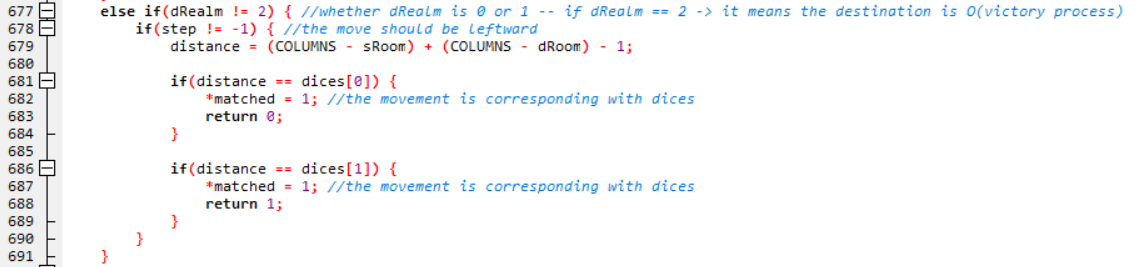
فقط به شرح قسمت‌هایی که شاید کمی مبهم باشند، می‌پردازیم.



این بخش‌ها بررسی می‌کنند که کدام تاس قبلاً استفاده شده‌است و اندیس عنصر تاس دیگر را برمی‌گردانند.



متغیر step بیانگر جهت حرکت مهره‌ها در قلمرویی است که در آن هستند.



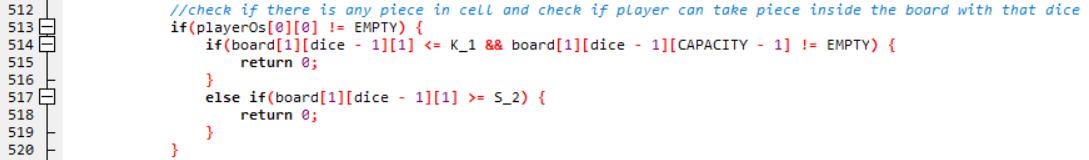
در صورتی که dRealm != 2 باشد، یعنی حرکت به جهت خارج کردن مهره نباشد، فاصله‌ی مبدأ با مقصد سنجیده شده و با تاس ها مطابقت داده می‌شود.

وظیفه : بررسی حرکات باقی‌مانده.

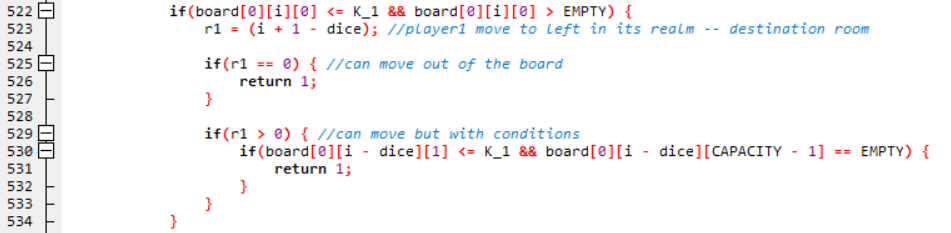
ورودی : متغیر صحیح player(صفر۰ برای بازیکن۱ و ۱ برای بازیکن۲) و متغیر صحیح dice که بیانگر مقدار تاس است.

خروجی : مقدار ۱ برای حالتی که بازیکن می‌تواند حرکت کند و مقدار صفر۰ برای حالتی که نمی‌تواند.

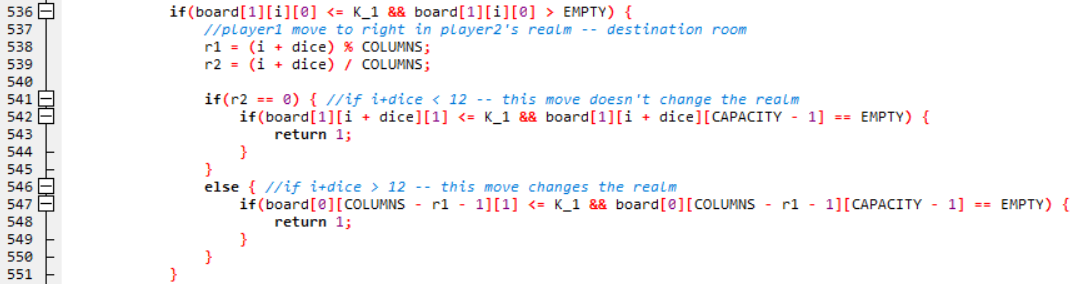
این تابع به دو بخش کلی مشابه یکدیگر تقسیم می‌شود. هر بخش مخصوص هر بازیکن است که یکی از آن‌ها را مورد بررسی قرار می‌دهیم.



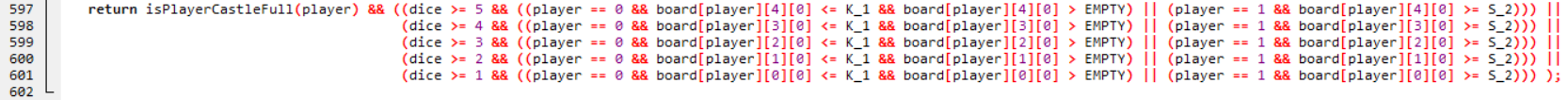
این بخش بررسی می‌کند که آیا بازیکن، مهره‌ای در زندان دارد یا خیر. در صورت مثبت بودن پاسخ، باید این مهره‌ی زندانی در صورت وجود فضا به قلعه‌ی بازیکن حریف منتقل شود، اگر فضایی برای انتقال آن مهره نبود، بازیکن تا مهره‌ای در زندان دارد نمی‌تواند از تاس های خود برای حرکت سایر مهره‌ها استفاده کند.



این بخش به حرکت مهره‌های بازیکن در قلمرو خودش می‌پردازد. اگر مهره‌ی بازیکن یافت شد می‌تواند در صورت باقی‌بودن بقیه‌ی شرایط(پر بودن قلعه) آن را به بیرون منتقل کند یا آن را اگر قوانین بازی(وجود مهره‌های خودش در خانه‌ی مقصد یا خانه‌ی خالی مقصد و ...) اجازه دهند به خانه‌ای دیگر منتقل کند.



این بخش هم مشابه بالا به حرکت بازیکن در قلمرو بازیکن حریف می‌پردازد.



هدف از این return، اجرای قانون ۱۰ بازی است.

وظیفه : حرکت دادن مهره‌ها.

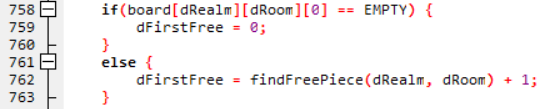
ورودی : متغیر صحیح player(صفر۰ برای بازیکن۱ و ۱ برای بازیکن۲)، متغیر صحیح dice که بیانگر عدد تاس می‌باشد. متغیرهای صحیح sRealm، sRoom، dRealm، dRoom که به ترتیب نمایانگر قلمرو مبدأ، ستون مبدأ، قلمرو مقصد، ستون مقصد هستند.

خروجی : مقدار WRONG\_MOVE برای نادرستی حرکت و مقدار DONE در صورت انجام شدن درست فرآیند.

فقط به شرح قسمت‌هایی که شاید کمی مبهم باشند، می‌پردازیم.



متغیر nextPlayer بیانگر اندیس بازیکن حریف است.



در صورتی که ستون مقصد خالی باشد، مهره باید ردیف اول آن قرار گیرد و در صورتی که حاوی مهره باشد، باید روی روترین ردیف پس از مهره‌ها قرار گیرد.

وظیفه : مشخص کردن وضعیت پربودن قلعه‌ی بازیکن(جمع شدن همه‌ی مهره‌های آن بازیکن در خانه‌های قلعه‌اش).

ورودی : متغیر صحیح player(صفر۰ برای بازیکن۱ و ۱ برای بازیکن۲).

خروجی : متغیر صحیح flag که بیانگر درستی یا نادرستی پربودن قلعه‌ی بازیکن است.

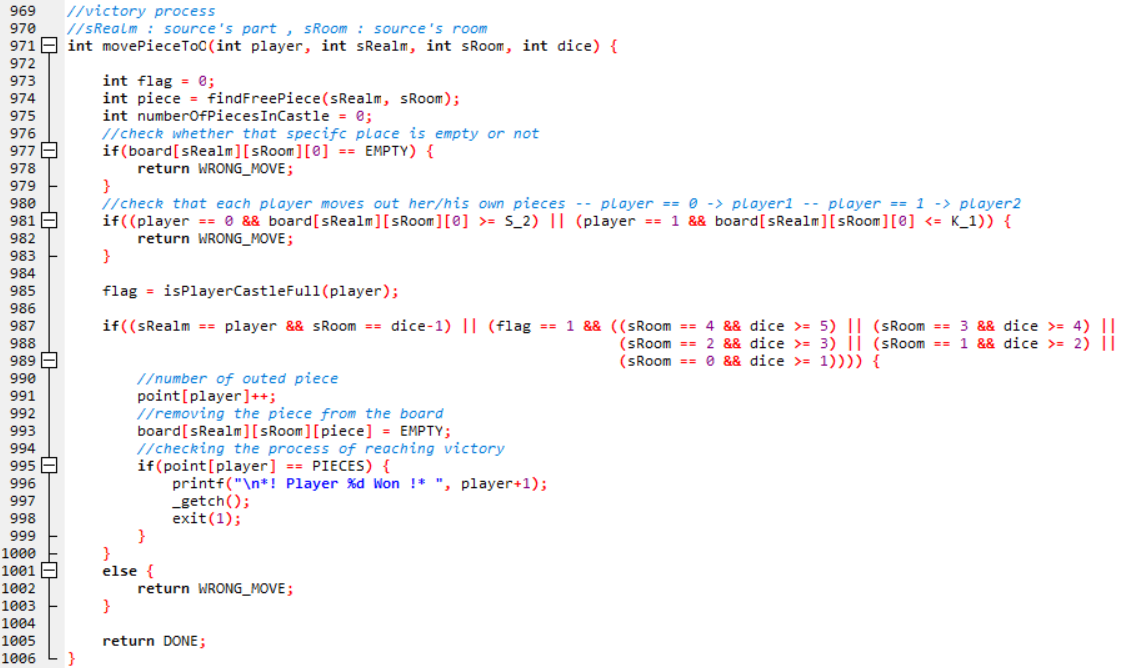


توسط حلقه‌های تودرتوی داخل تابع، به دنبال مهره‌های هر بازیکن در خانه ‌های قلعه‌اش می‌گردیم و با هر بار یافت شدن آن‌ها، یک واحد به متغیر numberOfPiecesInCastle افزوده می‌شود. سپس در پایان بررسی می‌کنیم که آیا این تعداد برابر با تعداد کل مهره‌ها منهای مهره‌هایی که تاکنون خارج شده‌اند، هست یا خیر. در صورت مثبت بودن پاسخ متغیر flag با مقدار ۱ بازگردانی می‌شود و در غیر این صورت با مقدار صفر۰.

وظیفه : انتقال مهره‌ها به خارج از بازی در فرآیند پیروزی.

ورودی : متغیر صحیح player(صفر۰ برای بازیکن۱ و ۱ برای بازیکن۲)، متغیر صحیح sRealm که بیانگر قلمرو مبدأ می‌باشد، متغیر صحیح sRoom که بیانگر ستون مبدأ است و متغیر صحیح dice که مقدار تاس را در خود دارد.

خروجی : مقدار WRONG\_MOVE برای نادرستی حرکت و مقدار DONE در صورت انجام شدن درست فرآیند.



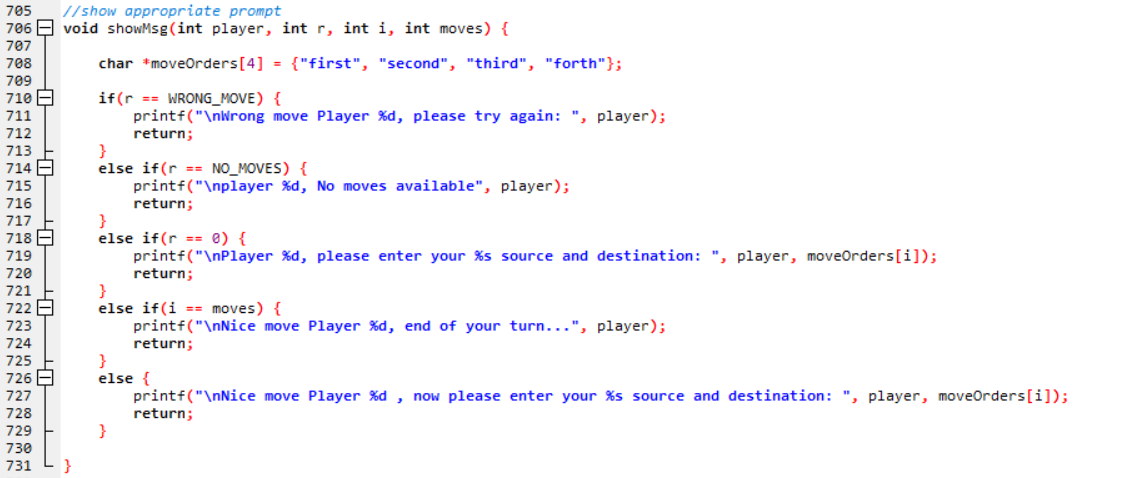
ابتدا مشخص می‌شود که مبدأ خالی از مهره نباشد و همچنین هر بازیکن مهره‌ی خود را حرکت دهد. سپس پربودن یا نبودن قلعه را مشخص می‌کنیم. سپس در صورت درست بودن قلمرو و مطابفت ستون با تاس، مهره ای بازی خارج شده و یک واحد به امتیاز بازیکن افزوده می‌شود. در صورتی که بازیکن، همه‌ی مهره‌های خود را خارج کند، پیروز بازی شده و بازی به پایان می‌رسد.

خطوط ۹۸۷ تا ۹۸۹ : شرط چند قسمتی داده شده برای اجرای قانون شماره ۱۰ بازی است.

وظیفه : چاپ پیام خروجی مناسب

ورودی : متغیر صحیح player(صفر۰ برای بازیکن۱ و ۱ برای بازیکن۲)، متغیر صحیح r(یک نوع response)، متغیر صحیح i و moves که برای مقایسه‌ی درستی تعداد دفعات حرکت بازیکن هستند.

خروجی : ندارد.

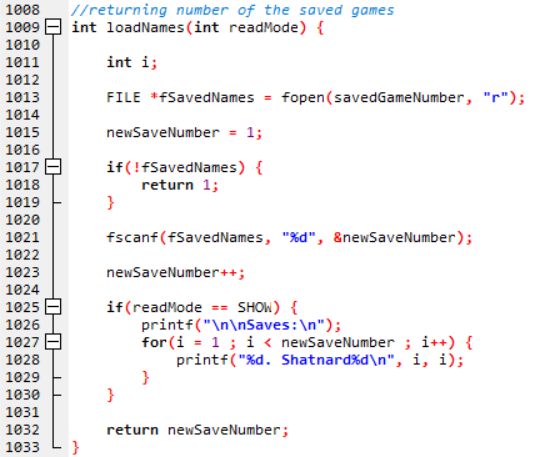


خط ۷۰۸ : دلیل ایجاد این آرایه از اشاره‌گر ها، کاربرپسند‌تر کردن بازی و مشخص کردن این است که بازیکن در حال انجام چندمین حرکت خود است.

وظیفه : شماره‌گذاری فایل های ذخیره‌شده به ترتیب و بازگردانی شماره‌ی جدید.

ورودی : متغیر صحیح readMode که بیانگر حالتی است که بخواهیم فایل‌های ذخیره‌شده را نشان دهیم.

خروجی : متغیر صحیح newSaveNumber که شماره جدید فایل‌های ذخیره شده است.

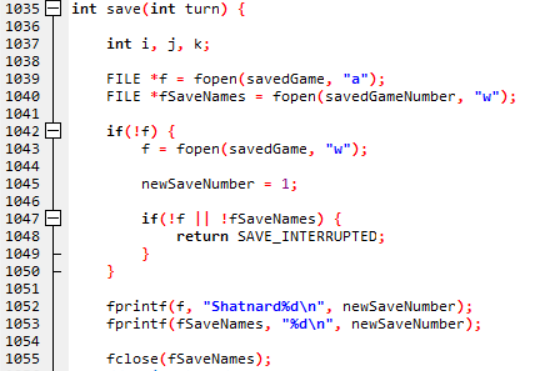


ابتدا فایل .dat ذخیره شده(numberOfSavedGames) برای خواندن(پرچم r) باز می‌شود. سپس شماره جدید از فایل fSavedNames خوانده می‌شود و یک واحد به آن افزوده می‌شود. سپس در صورتی که کاربر قصد ادامه‌ی بازی از فایل‌های ذخیره‌شده را داشته باشد، متغیر readMode برابر ثابت SHOW شده و نام فایل‌های ذخیره شده نشان داده می‌شود.

وظیفه : ذخیره سازی.

ورودی : متغیر صحیح turn که بیانگر شماره‌ نوبت بازیکنی است که در حال حرکت است.

خروجی : مقدار SAVE\_INTERRUPTED برای نادرستی فرآیند ذخیره‌سازی و مقدار DONE برای درستی.



در ابتدا فایل ذخیره‌ی بازی برای نوشتن در انتهای آن(پرچم a) و سپس فایل حاوی شماره‌ی ذخیره‌سازی برای نوشتن(پرچم w) گشوده می‌شوند. در صورتی که اولین بار باشد که بازی قرار است ذخیره شود، فایل ذخیره‌ی بازی(gameSave) برای نوشتن(پرچم w) باز می‌شود. سپس نام ذخیره‌سازی با فرمت گفته شده در قوانین در فایل‌ها نوشته می‌شود.



برای ذخیره‌ی بازی نیز اطلاعات مورد نیاز شامل موارد زیر به ترتیب در فایل ذخیره‌ی بازی بارگذاری می‌شوند: صفحه‌ی بازی، مهره‌های زندانی، مهره‌های خارج شده، چاله ها، سکه ها، ممنوعیت بازیکن ها از حرکت، سکه های جمع شده توسط هر بازیکن و درنهایت نوبت بازی. سپس فایل بسته شده و مقدار DONE بازگردانده می‌شود.

برخی نکات مهم :

* + با توجه به داده‌های ورودی، پیام مناسب چاپ می‌شود.
  + پس از تمام شدن حرکات یک بازیکن، پرسیده می‌شود که آیا کاربر مایل است بازی را ذخیره‌کند یا به سراغ ادامه‌ی بازی رود.
  + وجود توابع getch()‌ در بسیاری از قسمت‌ها، برای این است که زمان کافی برای نمایش پیام ها وجود داشته‌باشد.
  + در صورت نادرست بودن یک ورودی، دوباره از کاربر خواسته می‌شود که ورودی خود را تایپ کند.