

Snake Project

Mahdi Baziyar | Advanced Programing | July 2019

هدف:

ایجاد بازی کلاسیک Snake در قالب زبان برنامه نویسی جاوا.

این بازی شامل یک برد با ابعاد تعیین شده است که مار در آن با کلید های جهتی حرکت کرده و در صورت برخورد با موانع موجود و یا دیواره برد، به کاربر پیغام Game Over را نشان داده و متوقف می گردد. در این بازی برای رشد مار از سیبی که به صورت تصادفی در برد بازی ایجاد می شود استفاده نموده ایم. با هربار برخورد با سیب، بسته به کد نوشته شده، یک یا چند عدد به طول مار اضافه می شود. در برخی مراحل بازی قابلیت تلپورت از نقطه ای مشخص به نقطه ای دیگر نیز قرار داده شده است.

مراحل بازی شامل:

مرحله یک: شامل بردی است بدون هر گونه مانع، که مار با برخورد به دیواره های بازی نیز Game Over نشده و از سمت دیگر برد بازی ظاهر می گردد. در این برد، مار سه واحد طول دارد و با هربار ظاهر شدن سیب سبز در برد بازی و برخورد با آن، یک واحد به طول مار اضافه می شود.

مرحله دو: شامل بردی است مانند برد قبلی، با این تفاوت که مار با برخورد به دیواره های بازی Game Over خواهد شد و قابلیت تلپورت به سمت دیگر مسیر را از دست داده است. در این برد، همانند برد پیشین، مار سه واحد طول دارد و با هربار ظاهر شدن سیب سبز در برد بازی و برخورد با آن، یک واحد به طول مار اضافه می شود.

مرحله سه، بخش اول: شامل بردی است مانند برد قبلی، مار با برخورد به دیواره های بازی Game Over خواهد شد و قابلیت تلپورت به سمت دیگر مسیر را از دست داده است. تفاوت این مرحله با مرحله پیشین، موانعی است که در زمین بازی قرار گرفته و مار با برخورد به آن ها باعث Game Over شدن بازیکن می گردد. در برد فعلی مختصاتی از یک فایل متنی که در پوشه بازی قرار دارد خوانده شده و موانع با استفاده از آن ها، در زمین بازی ایجاد می شوند. در این برد، همانند برد پیشین، مار سه واحد طول دارد و با هربار ظاهر شدن سیب سبز در برد بازی و برخورد با آن، یک واحد به طول مار اضافه می شود.

مرحله سه، بخش دوم: شامل بردی است مانند برد قبلی، مار با برخورد به دیواره های بازی Game Over خواهد شد و قابلیت تلپورت به سمت دیگر مسیر را از دست داده است. تفاوت این مرحله با مرحله پیشین، موانعی است که در زمین بازی قرار گرفته و مار با برخورد به آن ها باعث Game Over شدن بازیکن می گردد. در برد فعلی تعدادی مشخصی مانع به صورت تصادفی در زمین بازی ایجاد می شوند. در این برد، همانند برد پیشین، مار سه واحد طول دارد و با هربار ظاهر شدن سیب سبز در برد بازی و برخورد با آن، یک واحد به طول مار اضافه می شود.

مرحله چهارم: شامل بردی است مانند برد قبلی، مار با برخورد به دیواره های بازی Game Over خواهد شد و قابلیت تلپورت به سمت دیگر مسیر را از دست داده است. تفاوت این مرحله با مرحله پیشین، تعداد واحد های افزوده شده به طول بدن مار با هر بار برخورد با سیب است. همانگونه که گفتیم، در مراحل پیشین با هر بار برخورد مار به سیب سبز، یک واحد به طول آن اضافه می شد؛ در این مرحله اما با تغییر در مشخصات سیب، کاری می کنیم که با هر بار برخورد مار با سیب طلایی به جای یک واحد، دو واحد به طول آن افزوده شود. در این برد همانند برد پیشین، مار سه واحد طول دارد.

مرحله پنجم: شامل بردی است مانند برد قبلی، مار با برخورد به دیواره های بازی Game Over خواهد شد و قابلیت تلپورت به سمت دیگر مسیر را از دست داده است. تفاوت این مرحله با مرحله پیشین، دو خانه با مشخصات خاص در برد بازی است. در این برد، دو خانه از زمین بازی با علامت ضربدر قرمز مشخص شده است. این دو خانه در مختصاتی تصادفی از زمین بازی ساخته می شوند. با هر بار برخورد سر مار با یکی از این خانه ها، بدن مار از خانه بعدی بیرون می آید یا به عبارت دیگر، مار به خانه دیگر با ضربدر قرمز تلپورت می شود. در این برد، همانند برد پیشین مار سه واحد طول دارد و بر خلاف آن هربار ظاهر شدن سیب سبز در برد بازی و برخورد سر مار با آن، یک واحد به طول مار اضافه می شود. مانند قبل در این برد نیز تعدادی مشخصی مانع به صورت تصادفی در زمین بازی ایجاد می شوند.

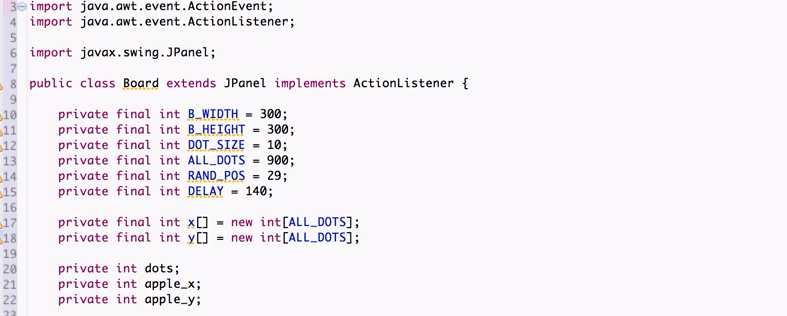
مرحله ششم: ترکیب مراحل چهارم و پنجم است. شامل بردی است مانند برد قبلی، مار با برخورد به دیواره های بازی Game Over خواهد شد و قابلیت تلپورت به سمت دیگر مسیر را از دست داده است. دو خانه از زمین بازی با علامت ضربدر قرمز مشخص شده است. این دو خانه در مختصاتی تصادفی از زمین بازی ساخته می شوند. با هر بار برخورد سر مار با یکی از این خانه ها، بدن مار از خانه بعدی بیرون می آید یا به عبارت دیگر، مار به خانه دیگر با ضربدر قرمز تلپورت می شود. در این برد، همانند برد پیشین مار سه واحد طول دارد و بر خلاف آن هربار ظاهر شدن سیب سبز در برد بازی و برخورد سر مار با آن، دو واحد به طول مار اضافه می شود. مانند قبل در این برد نیز تعدادی مشخصی مانع به صورت تصادفی در زمین بازی ایجاد می شوند.

شیوه پیاده سازی بازی:

در ابتدا برد اصلی بازی که دیگر مراحل با ارث بری از آن ساخته شده اند را ایجاد می کنیم.

در IDE، کلاسی با نام Board ایجاد می کنیم. این کلاس از JPanel ارث بری می کند و ActionListener را implement می کند. از JPanel برای ایجاد برد بازی و دیگر مشخصه های آن استفاده می شود. از ActionListener برای افزودن قابلیت حرکت با کلید های جهتی استفاده می کنیم.

در ادامه از اسکرین شات هایی از برنامه در مراحل مختلف برای توضیح بخش های مختلف کلاس Board استفاده می کنیم. گفتنی است که با پیشروی در توسعه برنامه، شماره خط کد ها عوض می شود.



در خط های 3 و 4 دو پکیج مربوط به ActionListener که ActionEvent و ActionListener باشند، import می شوند. این دو برای گرفتن دستورات از ورودی و در نهایت انجام عملی دلخواه پس از دریافت ورودی استفاده می شود. این دو به خاطر implement نمودن اینترفیس ActionListener، import شده اند.

در خط ششم، می بینیم که با extend نمودن کلاس Board از JPanel، پکیج JPanel نیز به کلاس فعلی import می شود تا توانایی استفاده از قابلیت های آن را داشته باشیم.

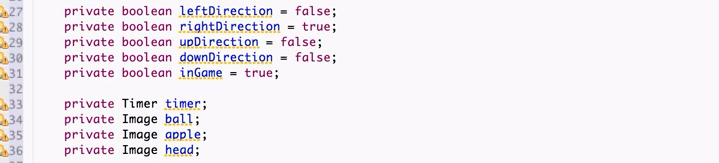
خط 8 شامل هدر کلاس Board است. Board کلاسی Public بوده که قابلیت های JPanel را به ارث برده و از توانایی های ActionListener نیز بهره می برد.

در خطوط 10 تا 15؛ دیتا فیلد هایی به صورت private و final برای ذخیره داده هایی مرتبط با بازی استفاده می کنیم. به این دلیل به صورت private تعریف شده اند که دیگر کلاس ها توانایی استفاده از این فیلد ها را نداشته باشند، به عبارتی با این کار سعی در پیاده سازی یکی از ویژگی های اساسی زبان های شی گرا به نام encapsulation داریم. این ویژگی به معنای کپسوله سازی و یا به عبارتی پنهان سازی بخش های مختلف کد ماست تا امنیت برنامه نهایی بیشتر شود و جز آن بخش از برنامه که ما به عنوان برنامه نویس تعریف می کنیم، دیگر بخش ها به مواردی که کپسوله سازی کرده ایم دسترسی نداشته باشند.

این دیتا فیلد ها شامل متغیر هایی برای ذخیره عرض و ارتفاع برد بازی، اندازه یک خانه از محیط بازی ( یک واحد از بدن مار، اندازه سیب و یا اندازه موانع و خانه ی تلپورت)، تعداد تمامی نقاط ممکن در زمین بازی که با ضرب عرض و ارتفاع برد بازی به دست آمده، ثابتی برای استفاده از متود Random که در نهایت برای ایجاد تصادفی مکان سیب، دیوار ها و یا خانه ی تلپورت استفاده می گردد. ثابت DELAY نیز سرعت بازی را مشخص می کند.

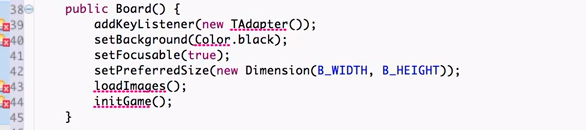
در خطوط 17 و 18، دو آرایه برای ذخیره مختصات x و y تمام خانه های بدن مار ایجاد نموده ایم.

در خطوط 20 تا 22، یک متغیر برای تعداد خانه های بدن مار و دو متغیر برای ذخیره مختصات سیب داریم.



در خطوط 27 تا 30، چهار متغیر Boolean برای مشخص نمودن وضعیت کلید های جهتی به کار می بریم. خط 31 برای مشخص نمودن بودن در بازی است.

در خطوط 33 تا 36، به ترتیب یک تایمر برای مشخص نمودن حرکت مار و سه تصویر برای سر، بدن و سیب تعریف می کنیم که بعدا در تابع loadImages() ، این تصاویر را به برنامه وارد می کنیم.

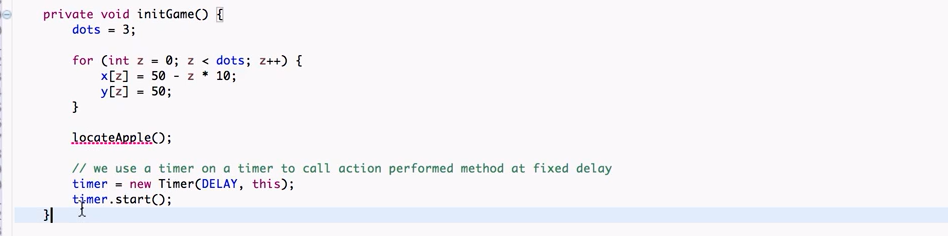


در خط 38، تابع سازنده کلاس Board را ایجاد می کنیم، وظیفه این تابع افزودن یک keyListener برای گرفتن ورودی از کلید های جهتی، مشخص نمودن رنگ پس زمینه برد بازی، در فوکوس قرار دادن برد بازی، تعیین اندازه برد بازی و در نهایت اجرای متود های loadImages() و initGame() است.



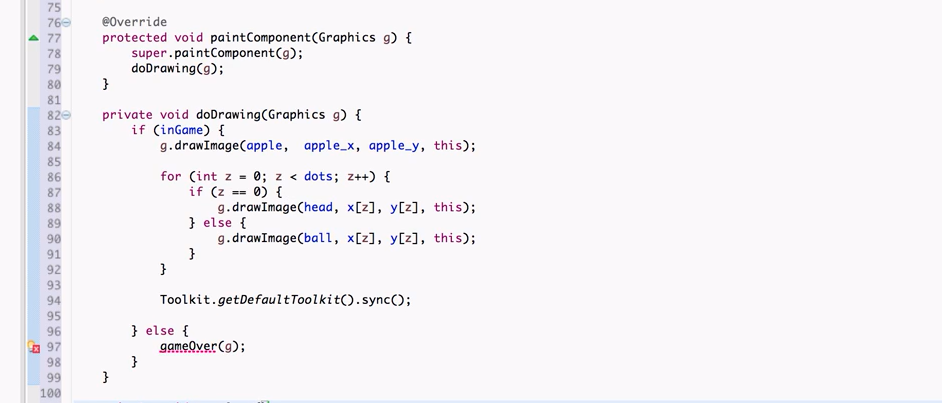
در متود loadImages() ما تصاویر لازم برای بخش های مختلف بازی را load می کنیم. تصویری برای بدن مار، تصویری برای سر مار و تصویری نیز برای سیب.

کلاس ImageIcon برای نمایش تصاویر با فرمت PNG به کار می رود. گفتنی است که باید تصاویری که در این پروژه استفاده می کنیم، همگی اندازه 10 در 10 پیکسل داشته باشند.



در متود initGame()، ما اندازه اولیه مار را مشخص می کنیم. در حلقه for ایجاد شده مکان ظاهر شدن مار را به صورت تصادفی ایجاد کرده و در نهایت با فراخوانی متود locateApple()، مکان سیب را به صورت تصادفی ایجاد می کنیم.

در خطوط بعدی ما از یک تایمر استفاده می کنیم تا متود actionPreformed با تاخیری ثابت (DELAY که در ابتدای برنامه تعریف کردیم) اجرا شود.



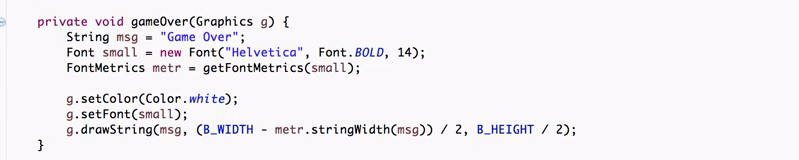
در خط 77، متود paintComponent که از JPanel به ارث رسیده را Override می کنیم، یعنی آن را طوری تغییر می دهیم که با نیاز های فعلی ما مطابق باشد. در این Override، ما ابتدا متود paintComponent کلاس JPanel را فراخوانی می کنیم و پس با فرستادن شی Graphics به متود doDrawing، وظیفه رسم سیب، سر و بدن مار را به آن می سپاریم.

paintComponent مربوط به Java Swing می باشد و زمانی فراخوانی می شود که می خواهیم گرافیک مورد نظر را برروی صفحه نمایش دهیم. آرگومان ورودی paintComponent یک شی از Graphics است که از java.awt.Graphics به دست آمده. پارامتر g یک شی از Graphics است. g در حقیقت یه ابزار رسم است، ما رنگ، شکل، اندازه و ... آن چه می خواهیم را مشخص می کنیم و دستور رسم آن را با متود paintComponet می دهیم.

در متود doDrawing، اعلام می کنیم که در صورت درحال اجرا بودن بازی، سیب در مختصات تصادفی ایجاد شده رسم شود و سپس سر و بدن مار نیز پشت سر هم رسم شوند.

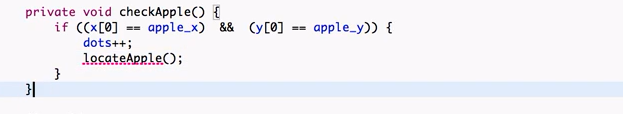
در نهایت در خط 94، ما از متود getDefaultToolkit() موجود در کلاس Toolkit استفاده می کنیم تا از نمونه toolkit ـی که در حال حاضر در برنامه استفاده می شود، با خبر شویم.

در خط 97، بیان می کنیم که در صورت در حال اجرا نبودن بازی (false بودن متغیر inGame)، متود gameOver اجرا شود.

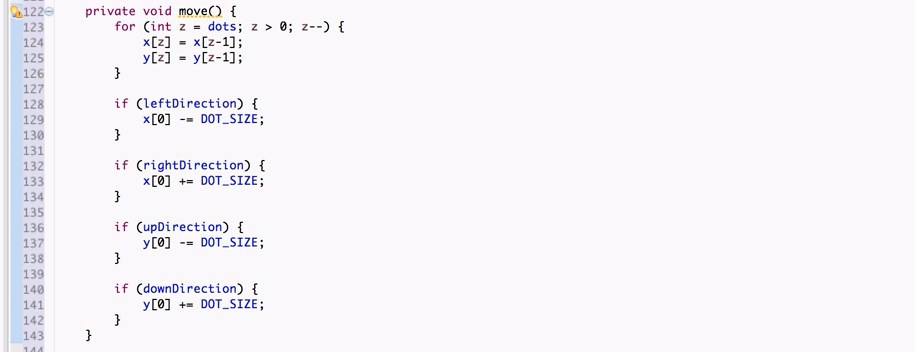


در متود gameOver که یک آرگومان ورودی از نوع Graphics دارد، مشخص می کنیم که در هر صورت که بازیکن در بازی نا موفق بود، صفحه ی پایان بازی نمایش داده شود.

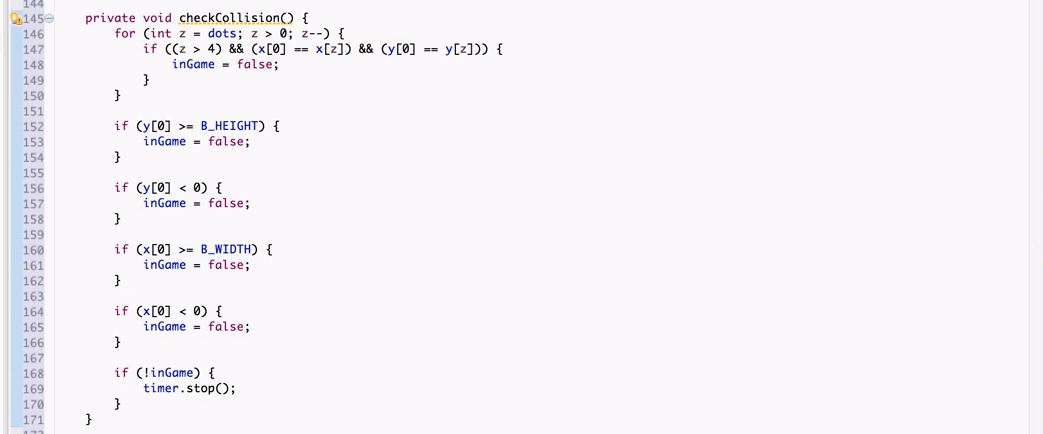
یک متن Game Over با فونت مشخص شده توسط شی ایجاد شده از کلاس Font که رنگ آن توسط کلاس Graphics تعیین می شود. سپس به همین طریق فرمان ایجاد این پیام در میانه صفحه بازی را می دهیم.



در متود checkApple، برخورد یا عدم برخورد مار با سیب را بررسی می کنیم. بدین منظور چک می کنیم که آیا مختصات سر مار با مختصات مار یکی است یا خیر. در صورت یکسان بودن دو مختصات یک واحد به طول بدن مار افزوده شده و سپس دوباره متود locateApple را برای ایجاد شدن سیبی دیگر در محیط بازی فراخوانی می کنیم.



در ادامه برنامه و خط 122، شاهد متود move هستیم. این متود همانگونه که از نامش بر می آید برای حرکت مار به کار می رود. تعدد اجرا شدن این متود به ثابت DELAY که در ابتدای برنامه تعریف کرده ایم وابسته است. در خط 123، شاهد تعریف شدن حلقه ای هستیم که کارش انتقال سر و بدن مار در طول زمین بازی است. در ادامه با چهار شرط مواجه می شویم که وظایف آن ها چک کردن فشرده شدن هر یک از کلید های جهتی و در نهایت انتقال مار در آن جهت است.

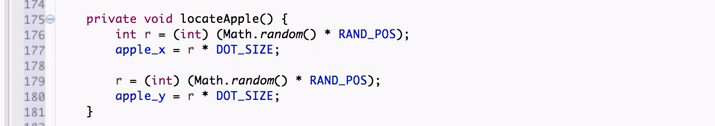


در بخش بالایی برنامه، متود checkCollision را داریم. وظیفه این متود چک کردن برخورد مار با بدن خود و چهار دیواره بازی است.

در خط 146 چک می کنیم که آیا سر مار با هر یک از بخش های بدن خود برخورد می کند یا خیر. در صورت برخورد متغیر inGame برابر false قرار داده شده و در نهایت بازی GameOver می شود.

در خطوط 152 تا 166، بررسی می شود که آیا سر مار با هر یک از دیواره های بازی برخورد دارد یا خیر. در صورت بیشتر و یا کمتر بودن هر یک از مختصات سر مار از ابعاد برد، متغیر inGame برابر با false قرار گرفته و در نتیجه، GameOver می شود.

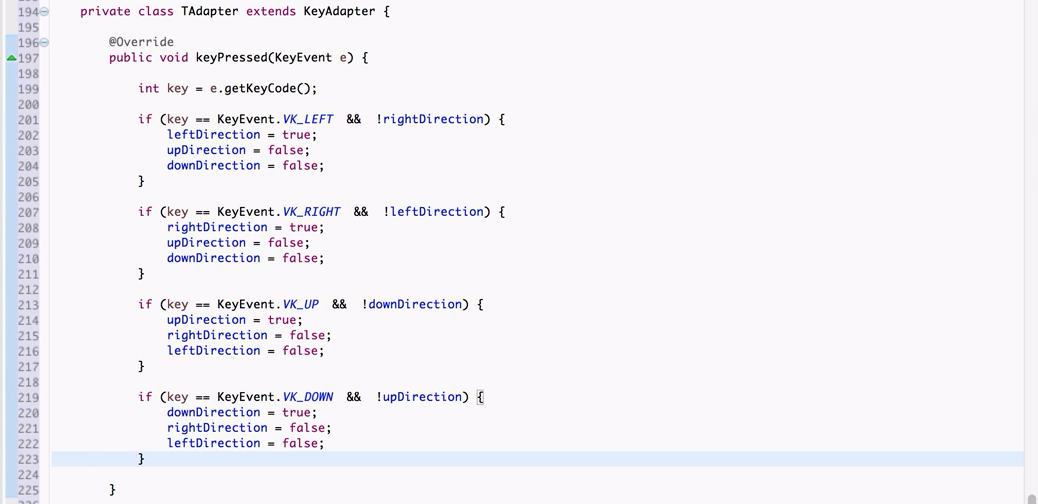
در خط 168، true یا false بودن متغیر inGame چک شده و در صورت false بودن، بازی متوقف می گردد.



در متود locateApple، مختصات x و y تشکیل سیب در زمین بازی را به صورت تصادفی و به کمک متود random در کلاس Math، می سازیم.

 در متود actionPreformed، معین می کنیم که با هر عملی که در بازی اتفاق می افتد، در صورت در حال اجرا بودن بازی، متود های checkApple،checkCollision و move فراخوانی شوند.

سپس با فراخوانی متود repaint که از روابط کلاس Graphics است، در خواست ریفرش شدن برد بازی را داریم.



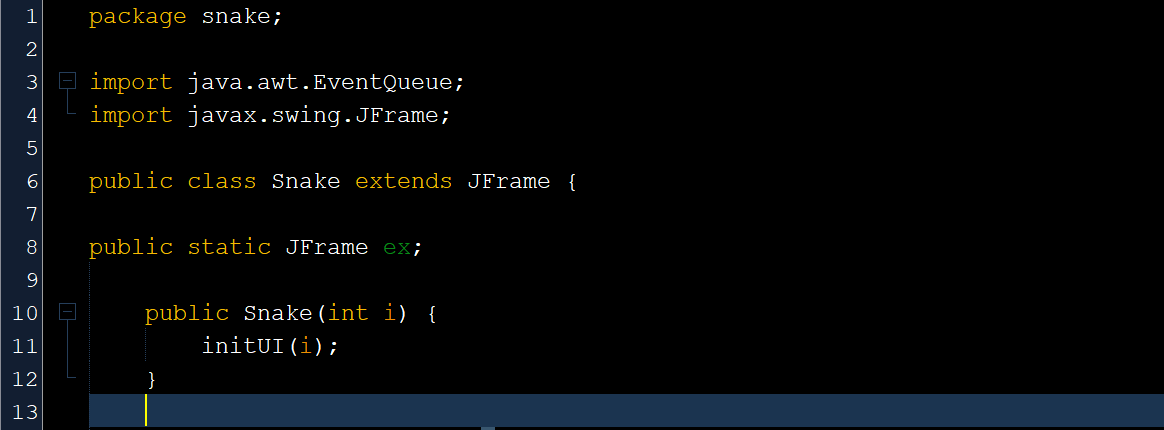
در ادامه، یک کلاس داخلی به نام TAdapter از کلاس abstract، keyAdapter به ارث می بریم.

در خط 197، متود keyPressed موجود در کلاس پدر را Override می کنیم و در بدنه متود ابتدا کد کلید فشار داده شده توسط کاربر را گرفته و در یک متغیر عددی قرار می دهیم.

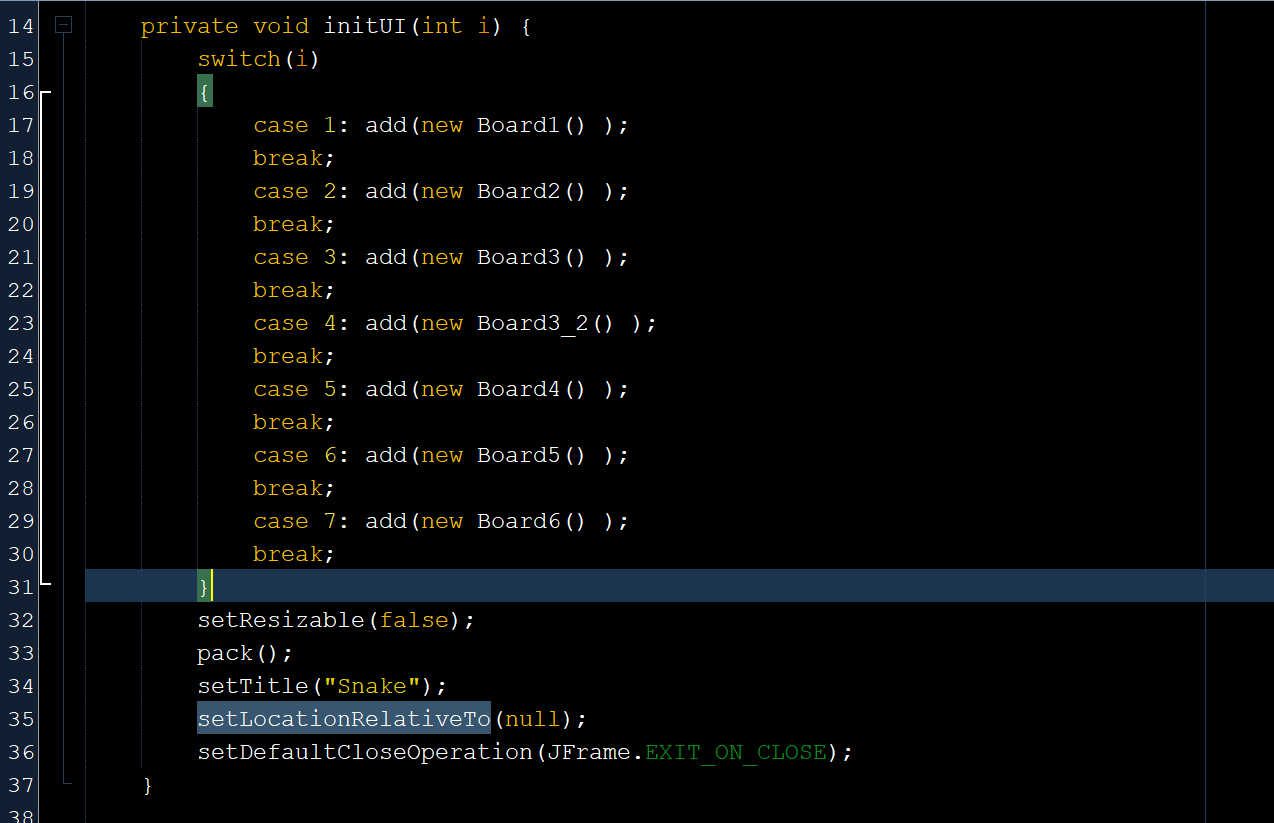
سپس با استفاده از چند شرط، چک می کنیم که آیا کلیدی که فشار داده شده یکی از چهار کلید جهتی هست یا خیر. در صورت درست بودن هر یک از شروط، متغیر Boolean مربوط به آن جهت برابر true قرار گرفته و دیگر متغیر ها false می شوند.

ایجاد کلاس Snake

در این بخش، ما مبادرت به ایجاد یک کلاس Snake برای مدیریت ایجاد صفحه بازی و ایجاد مراحل مختلف آن استفاده می کنیم.

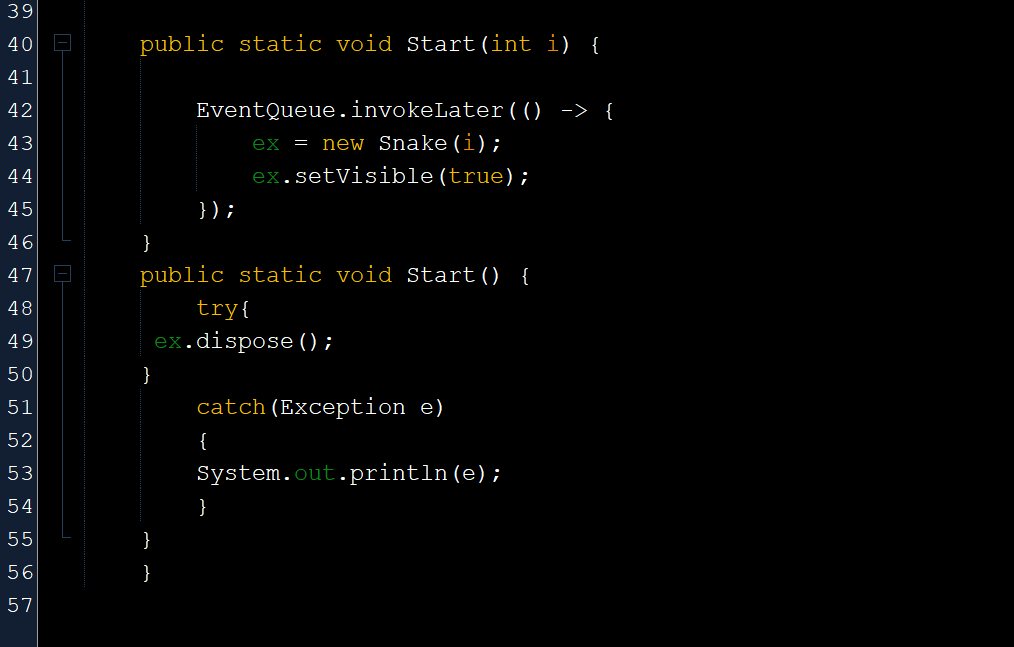


در این تصویر، در خط های 3 و 4 دو کلاس JFrame و EventQueue به برنامه import شده اند. خط 6 تعریف کلاس به نام Snake است که از JFrame به ارث می برد. در خط 8 یک فریم به نام ex تعریف نموده ایم. در خط 10 شاهد متود سازنده کلاس Snake هستیم که متغیری را دریافت و در نهایت به متود initUI می فرستد.



متود initUI یک آرگومان int دریافت می کند، در کد switch بررسی می کند که عدد دریافت شده با کدام یک از کیس ها برابر است و در نهایت بسته به عدد فرستاده شده، یک نمونه از یکی از مراحل بسازد و در فریم ex قرار دهد.

در خط 32 تا 36، ابتدا اجازه تغییر سایز پنجره به وسیله موس را از آن سلب کرده، سپس با متود pack()، می خواهیم که پنجره در ابعاد خاص قرار بگیرند. سپس عنوان و مکان صفحه را تنظیم نموده و در نهایت مشخص می کنیم که با کلیک بر علامت ضربدر بالای پنجره، برنامه متوقف شود.

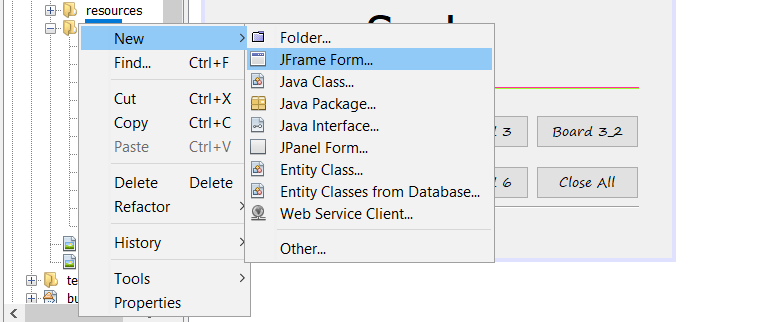


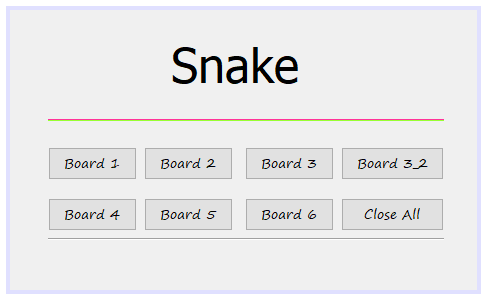
در خط 40 متود Start با یک عدد به عنوان آرگومان ورودی داریم. متود به این دلیل استاتیک تعریف شده تا نیاز به ساخت یک نمونه برای فراخوانی آن نداشته باشیم. این عدد از منو بازی ارسال می شود و برای ساخت بردی از مرحله انتخابی از منو استفاده می شود. در خط 42 دستور اجرا شدن یک سری از رویداد ها پشت سر هم هستیم. فریم ex را برابر با فریمی که در متود Snake ساخته می شود قرار می دهیم و سپس آن فریم را قابل مشاهده می کنیم.

در خط 47، متود Overload شده Start را داریم که آرگومان ورودی نداشته و برای بستن پنل در حال اجرا با فشردن دکمه close و با متود dispose() به کار می رود. این متود در بلاک try و catch قرار گرفته تا در صورت ساخته نشده بودن پنجره و فشردن دکمه توسط کاربر در منو، برنامه با اشکال مواجه نشود.

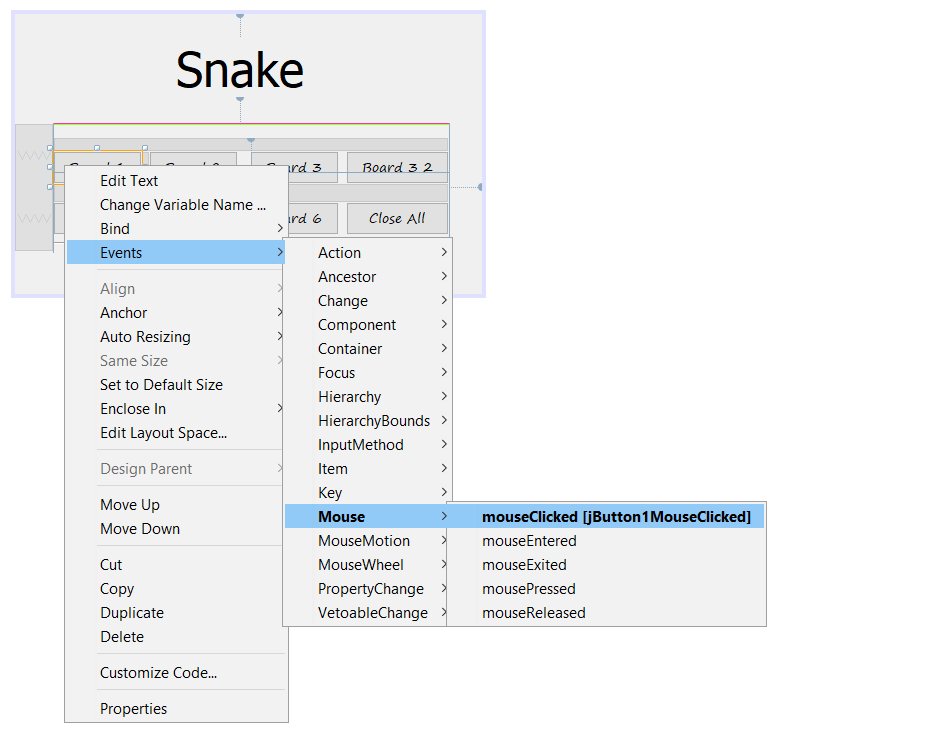
ایجاد منو

منوی بازی توسط ابزار های گرافیکی NetBeans و به صورت تصویر زیر ساخته شده است.

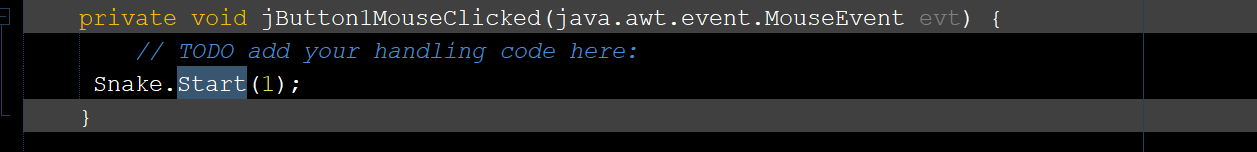




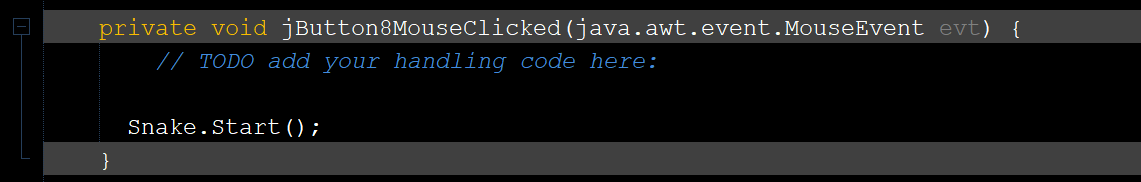
منوی بازی شمال 7 دکمه برای انتخاب مراحل مختلف بازی و یک دکمه برای بستن پنجره در حال اجرای بازی بدون متوقف کردن تمام برنامه است. یک JLabel برای نمایش عنوان Snake در بالای منو ایجاد کرده ایم، دو جدا کننده هم بالا و پایین دکمه ها قرار داده ایم.



به هر دکمه یک mouseListener افزوده ایم تا با کلیک کردن کاربر بر هر کدام از دکمه، دستورات دلخواه اجرا شوند. در این برنامه، دستورات ما شامل اجرای هر یک از مراحل بازی و بستن فریم در حال اجرا توسط دکمه Close All است.



برای مثال با کلیک بر روی دکمه Board 1، دستور Snake.Start(1) اجرا می شود. این دستور به این معناست که متود Start از کلاس Snake را با آرگومان ارسالی 1، اجرا کن. در ادامه در کلاس Snake، متغیر i برابر 1 قرار می گیرد و در نهایت با انتخاب شدن کیس شماره یک در switch، بورد مرحله یک بازی اجرا می شود.

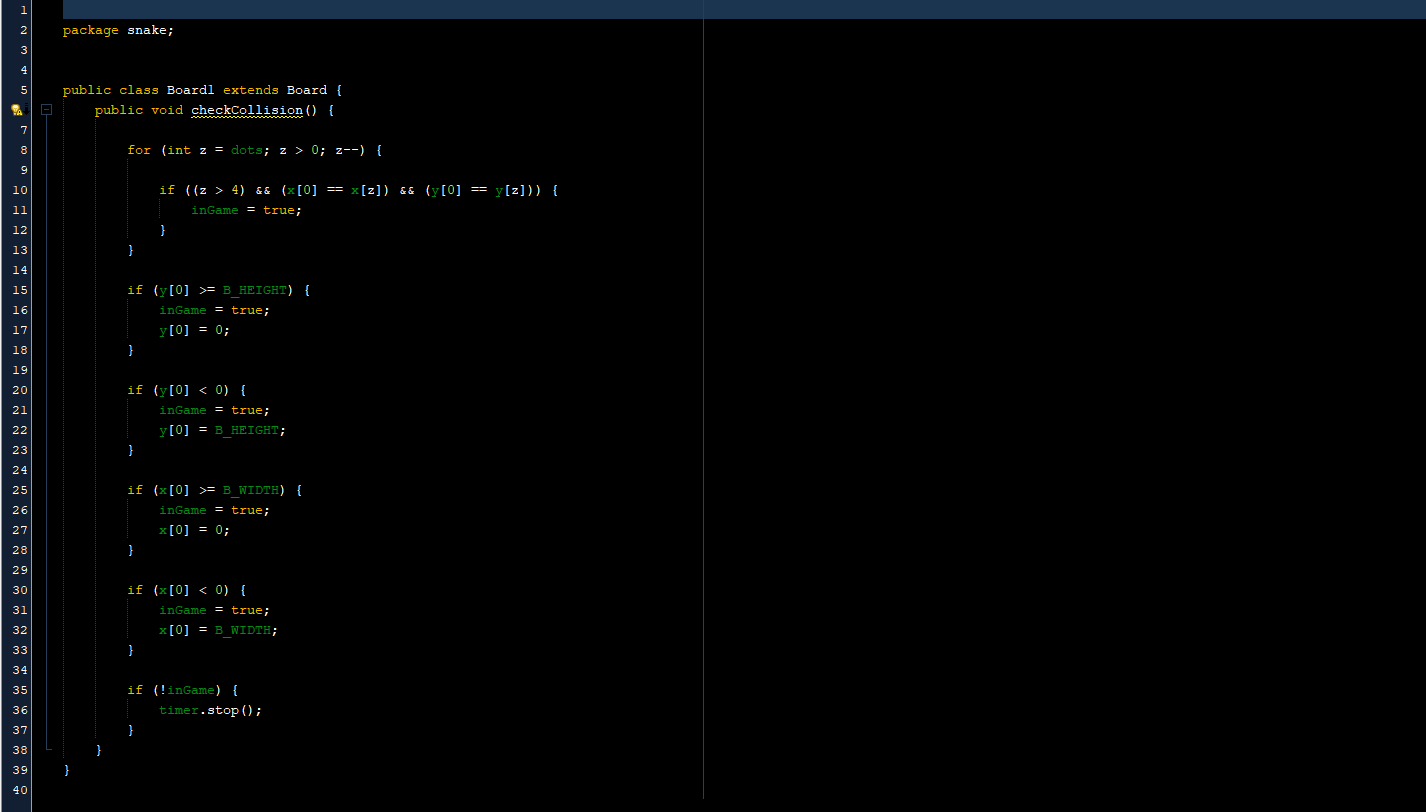


با فشردن کلید Close All، دستور Snake.Start() اجرا می شود که باعث بسته شدن هر پنجره ای از بازی Snake است.

شیوه ایجاد هر یک از شش مرحله بازی

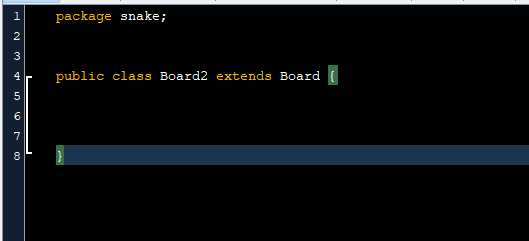
هر برد بازی با ایجاد تغییراتی در کلاس Board اصلی بازی ایجاد می شود. برای مثال برای ساخت مرحله اول بازی (کلاس Board1)، ما از کلاس Board تمام ویژگی های آن را به ارث برده ایم و سپس با بازنویسی و یا افزودن دستورات دلخواه، آن را به صورتی که میخواهیم تغییر می دهیم.

مرحله یک (Board1)



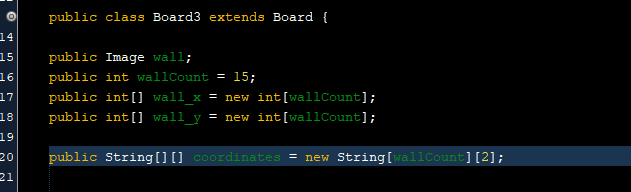
برای ساخت این برد از بازی، از کلاس Board ارث بری کرده و تنها متود checkCollision() آن را Override نموده ایم. در شرط های خطوط 15 تا 32، مشخص کرده ایم که هنگام برخورد مار با هر یک از چهار دیواره ی صفحه بازی، بجای Game Over شدن، مار از سمت دیگر صفحه بیرون بیاید. به عبارت دیگر تنها راه Game Over شدن در این مرحله برخورد سر مار با یکی از بخش های بدنش است.

مرحله دو (Board2)

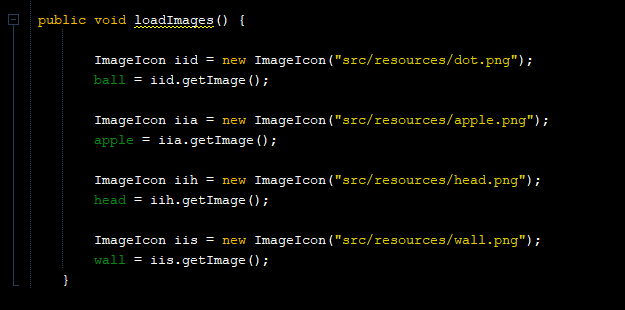


برای این مرحله تنها از کلاس Board ارث بری کرده ایم.

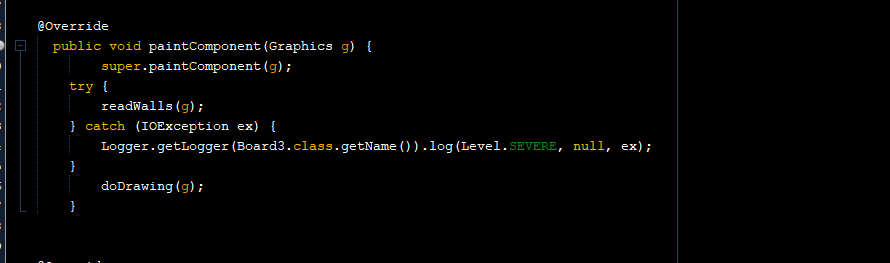
مرحله سه (Board3)



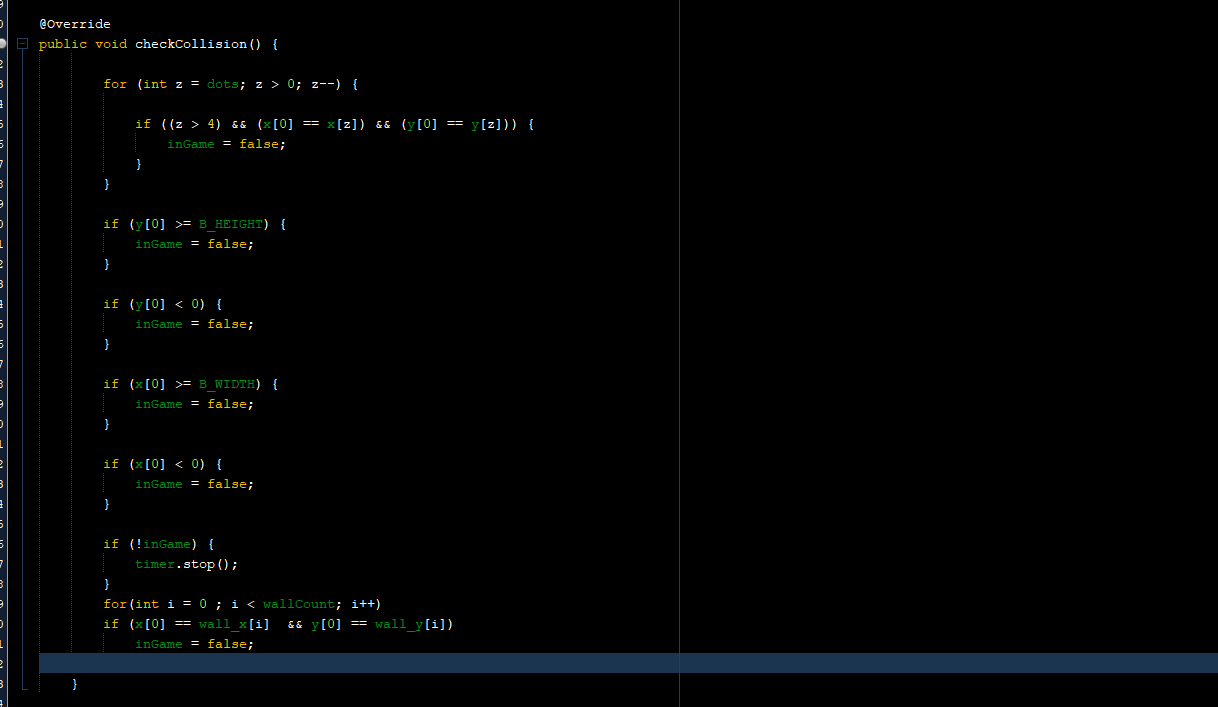
برای ایجاد این مرحله از کلاس Board ارث بری کرده ایم. یک تصویر برای دیوار تعریف می کنیم. سپس متغیری برای تعداد دیوار ها، دو آرایه عددی با تعداد خانه هایی به اندازه تعداد دیوارها برای ذخیره مختصات x و y هر یک از دیوار ها و در نهایت یک آرایه متنی دو بعدی ذخیره x و y خوانده شده از فایل ایجاد می کنیم.



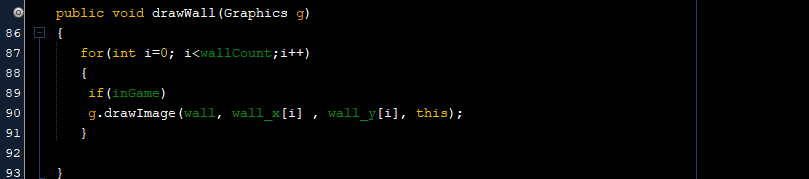
در ادامه تابع loadImages() را بازنویسی می کنیم و در آن تصویر مربوط به دیوار را از مکان مشخص شده می خوانیم.



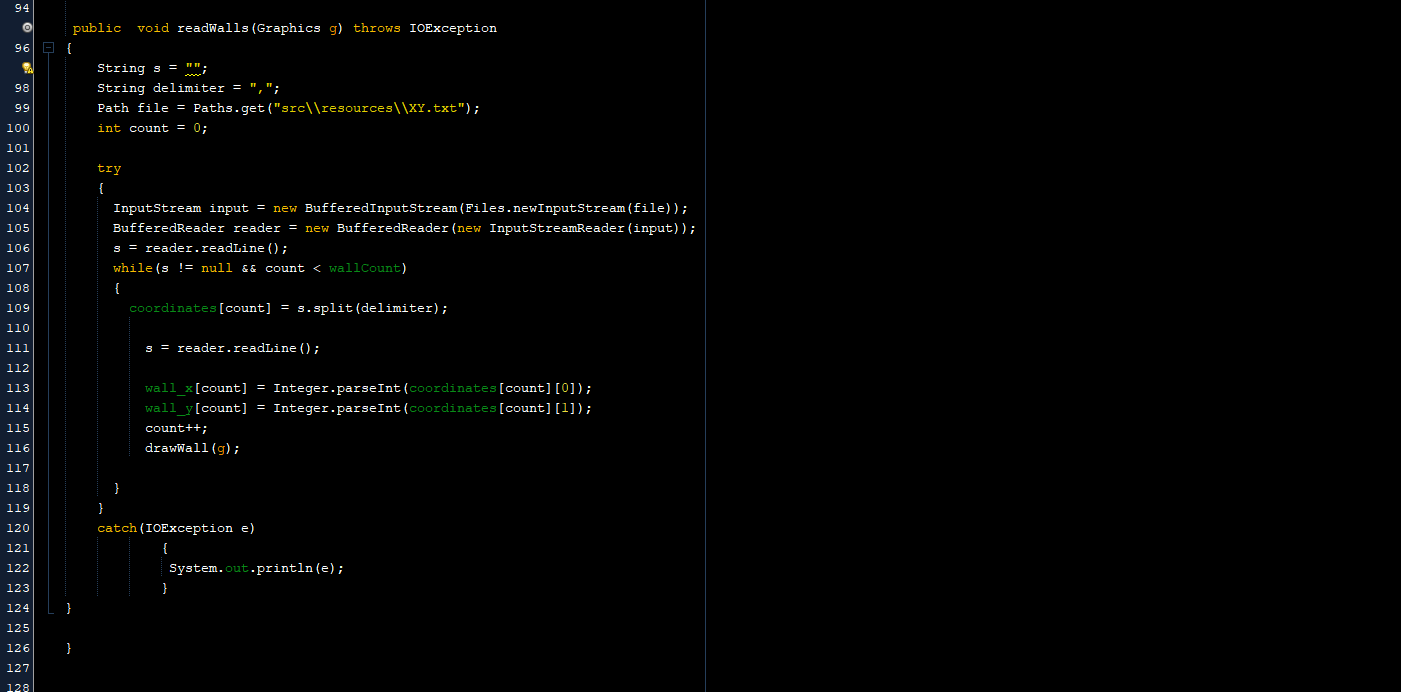
در مرحله بعد تابع paintComponent را بازنویسی می کنیم تا متود readWalls(g) را نیز با هربار اجرا شدن، فرخوانی کند. گفتنی است چون با فایل سر و کار داریم، این متود در بلاک try و catch قرار گرفته تا برنامه با مشکل مواجه نشود.



در ادامه باز متود checkCollision() را Override می کنیم و با قرار دادن یک حلقه در انتهای آن، وظیفه چک کردن برخورد سر مار با هر یک از موانع را به آن می سپاریم. اگر مختصات سر مار با مختصات هر کدام از دیوار ها برابر بود، بازی Game Over می شود.



در اینجا متود drawWall را داریم که وظیفه اش رسم دیوار ها به تعداد wallCount است.



در متود readWalls ما ابتدا خط به خط فایل XY.txt خوانده و سپس با جدا کردن مختصات فایل که با یک کاما از هم جدا شده اند، آن هارا در آرایه دو بعدی coordinates ذخیره می کنیم. سپس با تبدیل اعدادی که به صورت String در آرایه coordinates ذخیره شده اند به int، آن ها را در wall\_x و wall\_y می کنیم.

پس از خوانده شدن مختصات دیوار ها، متود drawWall را در خط 116 فراخوانی می کنیم تا دیوار ها را در برد بازی رسم نماید.