



PURE *MICHIGAN*®

Edition

Release 2.0

Professor: Dr. Nandigam

Class: CIS 350 Sec 01 Winter Semester

Team Members: Steve Meadows, Matt Ferretti, Isabel Albaitis

Table of Contents

Table of Contents	2
Project Description	3
Features Version 1	4
Features Version 2	4
End-Device Requirements	5
Version Control	5
Screen Shots	6
Use Case Diagram	12
Use Case Description	13
Play Game in Traditional Mode Use Case	13
Play Game in Time Trial Mode Use Case	14
View Game Instructions Use Case	16
UML CLASS DIAGRAM	17
Static Source Code Analyzers	18
SonarLint	18
Result of SonarLint	26
Code Coverage	27
Responsibilities & Reflections	33
Matt Ferretti	33
Steve Meadows	34
Isabel Albaitis	35

Project Description

This is a mobile application for android. The application is a memorization game where a single user is tasked to memorize the picture and the position of many different playing cards on the game board. Each playing card is faced down hiding their picture with the user only being able to select two cards at a time, revealing two pictures. Each card has an identical twin with the same picture but in a different location. If the user selects two cards with the identical picture then they will be awarded a match and those cards will disappear. If it's not a match those cards will flip back over hiding their picture, forcing the user to remember the positions of each picture. Eventually the user will find all matches and clear the board of cards. Once the board no longer contains any more cards and all matches are found the user will be presented with a screen displaying the time it took them to find all the matches. The goal is to find all matches on the board in the shortest possible time.

Features Version 1

- “Traditional” Mode
 - In this mode the clock will start at 0 and then start counting up. The user will match each card with its pair until there are no more pairs left and the board is empty. The goal is to match all pairs of cards as fast as possible
- The Difficulty of Traditional mode can be changed from either easy, medium, or hard with the initial difficulty set to easy.
 - Easy difficulty consists of 6 cards
 - Medium difficulty consists of 12 cards
 - Hard difficulty consists of 20 cards
- The user can view instructions on the game, what it is, what each difficulty is and how traditional mode is played

Features Version 2

- “Time Trial Mode”
 - In this mode the clock will start at a specific amount of time depending on the level being played and when the level starts the clock will start counting down. The player will have to race against the clock to find all matches. If all the matches are not found before time runs out, then the level will fail with a prompt to try again.
- Updated the instructions screen for the player to understand the game better and improved the finished screen to give the player a more rewarding sense of completion.
- A leveling system with levels from 1 all the way to 8, where the player can not advance to the next level before they complete the previous level. Below is a table showing the amount of cards each level along with the amount of time given in Time Trial mode to complete the level.

	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8
Number of cards	6	12	16	20	24	32	40	48
Time allowed	0:30	0:45	1:00	1:00	1:30	2:30	3:00	3:00

End-Device Requirements

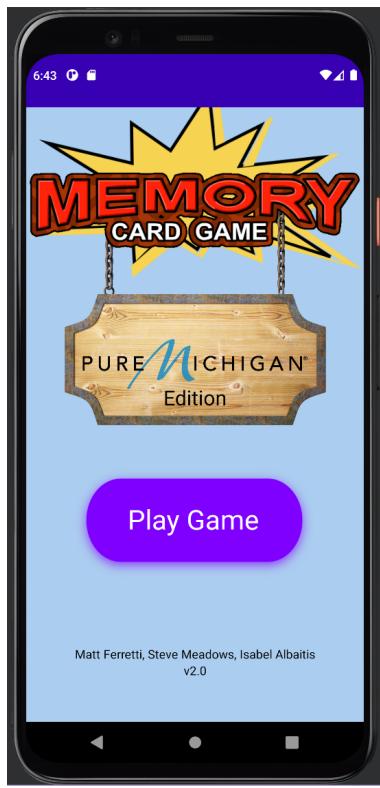
This application was developed and tested on a Google Pixel 5 API 30 running Android 11. It can be used on any Android phone running Android 30 or higher.

Version Control

The memory matching game is version controlled using Git and its repository is hosted on GitHub. The repository for this project is at the following site:

<https://github.com/steadows/Memory>

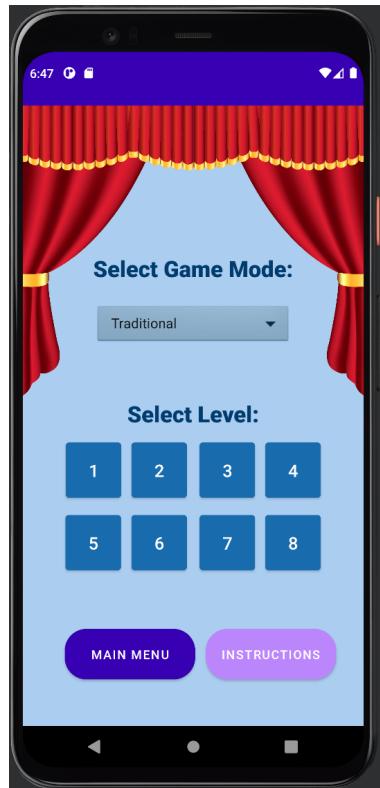
Screen Shots



Screen: Main Menu

Description:

This screen is the Main screen. Its function is to welcome the player to the app and entice them to play.



Screen: Main Menu

Description:

This screen is the Sub Menu screen. It allows the player to start the game, choose the mode, choose the level, look at the instructions, and return to the main menu.



Screen: Level One

Description:

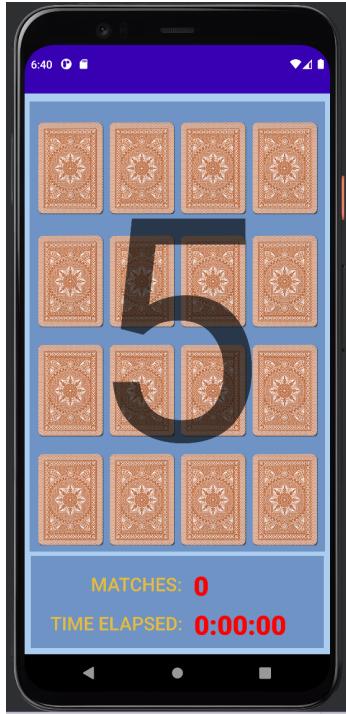
The player will try to match all 6 cards in the shortest time possible.



Screen: Level Two

Description:

The player will try to match all 12 cards in the shortest time possible.



Screen: Level Three

Description:

The player will try to match all 16 cards in the shortest time possible.



Screen: Level Four

Description:

The player will try to match all 20 cards in the shortest time possible.



Screen: Level Five

Description:

The player will try to match all 24 cards in the shortest time possible.



Screen: Level Six

Description:

The player will try to match 32 cards. Once 16 cards are matched the board will reset giving the player another 16 cards to match.



Screen: Level Seven

Description:

The player will try to match 40 cards. Once 20 cards are matched, the board will reset giving the player another 20 cards to match.



Screen: Level Eight

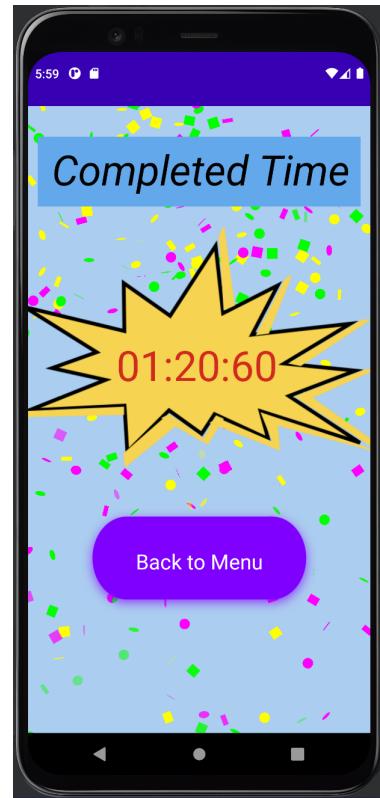
Description:

The player will try to match 48 cards. Once 24 cards are matched, the board will reset giving the player another 24 cards to match.

Completion Screens

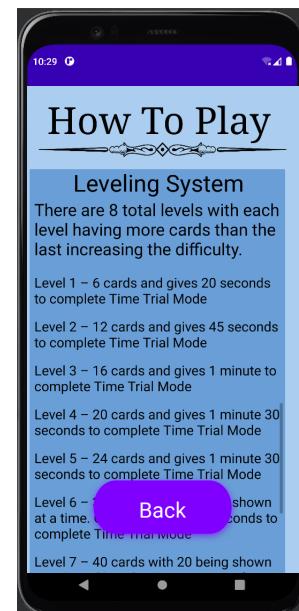
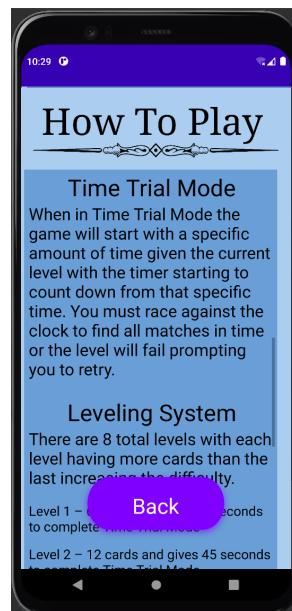
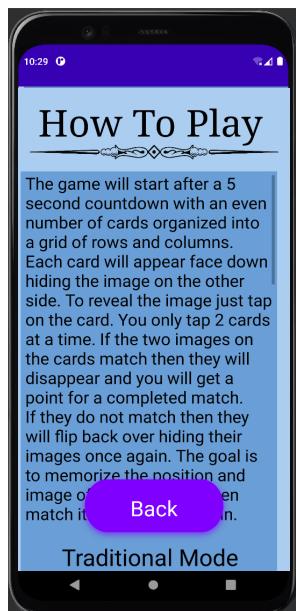


This screen shows when the player runs out of time in Time Trial Mode. They can either play the level again or return to the sub-menu



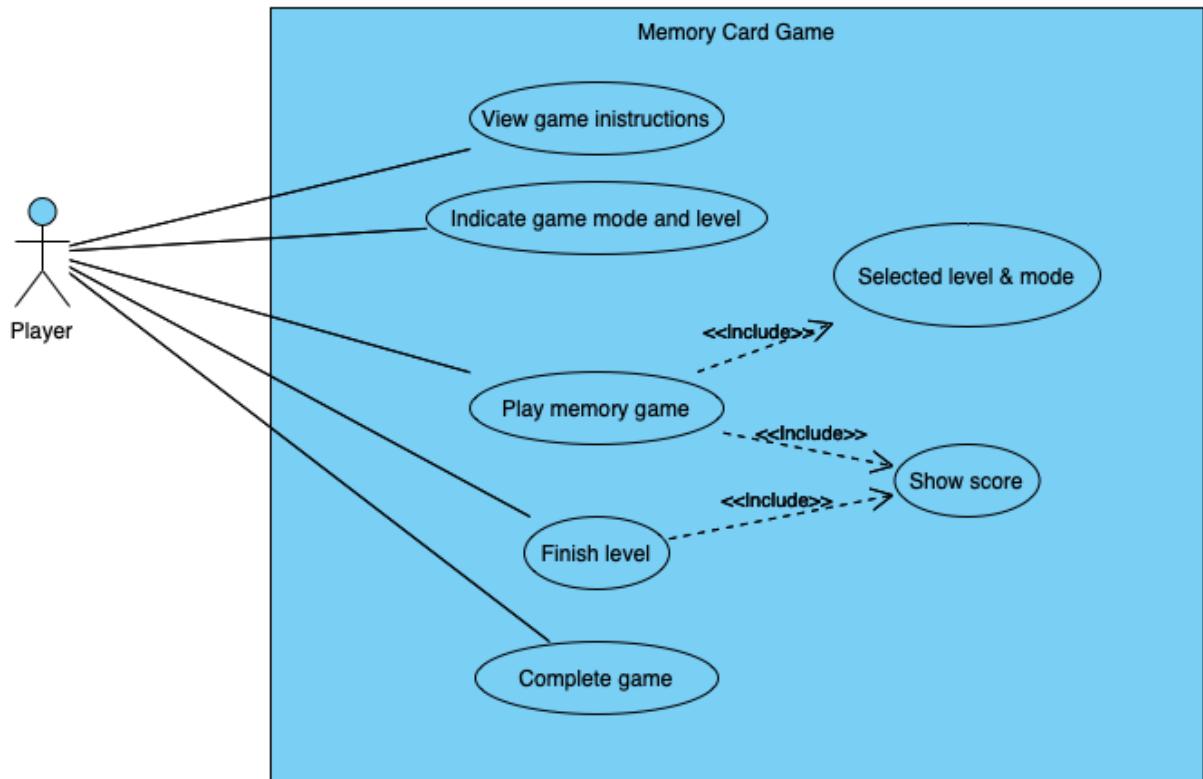
This screen shows when the player completes the level. It will either say "Completed time" or "Time Remaining" depending on the mode and show their respective time

Instruction Screen



The instructions screen tells the player how the game is played. It also explains both game modes and gives information about each level

Use Case Diagram



Use Case Description

Play Game in Traditional Mode Use Case

Name	Traditional Mode
ID	TR-MODE
Brief Description	Describes how the Player plays the Memory Game in traditional mode to complete all levels in traditional mode.
Actors	Player
Triggers	App Startup Player clicks “Play Game” button on main menu Player selects “Traditional Mode” from dropdown in submenu Player selects level to play in traditional mode
Preconditions	None
Primary Flow	<ol style="list-style-type: none">1. Player selects “Traditional Mode” from the dropdown in the submenu.2. Player selects the level to play.<ol style="list-style-type: none">a. The 5 second countdown timer begins.b. After the countdown timer is done, the Player can then begin selecting 2 cards at a time to collect two-of-a-kind matches.c. Gameplay time is shown in the “Time Elapsed” field.3. Player selects two cards at a time to collect matches.<ol style="list-style-type: none">a. If two card selections aren't a match, cards are flipped back over to the original state.b. If cards are a match, cards disappear from the board and the match counter is increased by one.4. Level ends when all cards are paired with matches.5. End screen loads showing time of completion<ol style="list-style-type: none">a. Player is prompted to return to the main menu.b. Level is marked as complete.6. Player unlocks the next level if available.

Alternate Flows	<p>Player selects locked level (Step 2): A Toast message will appear telling the user that the level they selected is locked. Player will remain on the submenu screen.</p>
Minimal Guarantees	App will continue to run until all matches are collected in each level.
Success Guarantees	Player makes every match on the board on every level in traditional mode.

Play Game in Time Trial Mode Use Case

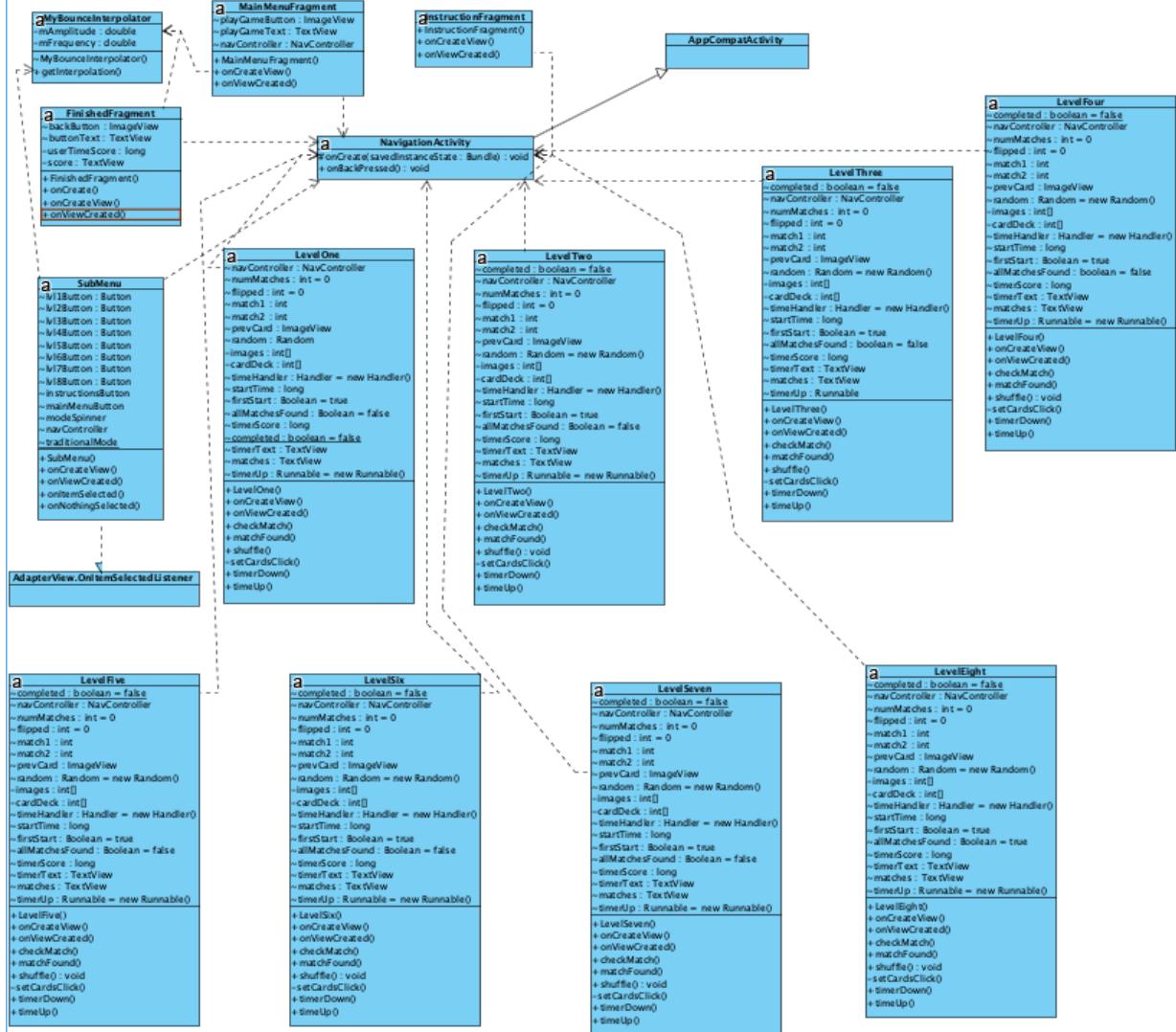
Name	Time Trial Mode
ID	TT-MODE
Brief Description	Describes how the Player plays the Memory Game in time trial mode to complete all levels in time trial mode.
Actors	Player
Triggers	App Startup Player clicks “Play Game” button on main menu Player selects “Time Trial Mode” from dropdown in submenu Player selects level to play in time trial mode
Preconditions	None

Primary Flow	<ol style="list-style-type: none"> 1. Player selects “Time Trial Mode” from the dropdown in the submenu. 2. Player selects the level to play. <ol style="list-style-type: none"> a. The 5 second countdown timer begins. b. After the countdown timer is done, the Player can then begin selecting 2 cards at a time to collect two-of-a-kind matches. c. Remaining time is shown in the “Time Remaining” field. 3. Player selects two cards at a time to collect matches. <ol style="list-style-type: none"> a. If two card selections aren't a match, cards are flipped back over to the original state. b. If cards are a match, cards disappear from the board and the match counter is increased by one. 4. Level ends when time runs out. 5. End screen loads showing the number of matches made by the Player in the time allotted. <ol style="list-style-type: none"> a. Player is prompted to return to the main menu. b. Level is marked as complete. 6. Player unlocks the next level if available.
Alternate Flows	<p>Player selects locked level (Step 2): A Toast message will appear telling the user that the level they selected is locked. Player will remain on the submenu screen.</p> <p>Player does not make all matches (Step 5): Player returns to the submenu. Level is not marked as complete.</p>
Minimal Guarantees	App will continue to run until all matches are collected in each level.
Success Guarantees	Player makes every match on the board on every level in time trial mode.

View Game Instructions Use Case

Name	Click Here to learn how to play
ID	InstructionFragment
Brief Description	Describes the rules of the game and how to interact with the application to win the game.
Actors	App User
Triggers	App Startup User selects 'Play Game' Button User selects 'Click Here to learn how to play'
Preconditions	None
Primary Flow	<ol style="list-style-type: none"> 1. User selects the 'Click Here to learn how to play' button. 2. Instructions display on screen. 3. Users can scroll to view the entire screen. 4. Instruction screen can be exited by selecting the 'back' button on the device.
Alternate Flows	None
Minimal Guarantees	Users can view the full menu.
Success Guarantees	Users can continue onto gameplay.

UML CLASS DIAGRAM



Static Source Code Analyzers

SonarLint

For app development using android studio as an IDE we found that the best code analyzer to use was Sonar Lint. By using SonarLint we were able to enforce uniform coding standards across the team allowing the code to be easily understood by everyone on the development team and anyone that will use it in the future. Conforming to SonarLint's coding standard also allows for the team to find potential bugs when adding new features and easily be able to fix them. All of the SonarLint coding standards that were used and checked for are below along with the level of significance indicator .



- Indicates Minor Bug



- Indicates Major Bug



- Indicates Critical Bug



- Indicates Information Blocking

".equals()" should not be used to test the values of "Atomic" classes	!
"=+" should not be used instead of "+="	!
"@CheckForNull" or "@Nullable" should not be used on primitive types	!
"@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects	!
"@Deprecated" code marked for removal should never be used	!
"@Deprecated" code should not be used	!
"@NonNull" values should not be set to null	!
"@Override" should be used on overriding and implementing methods	!
"@RequestMapping" methods should not be "private"	!
"@SpringBootApplication" and "@ComponentScan" should not be used in the default package	!
"ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization	!
"Arrays.stream" should be used for primitive arrays	!
"BigDecimal(double)" should not be used	!
"Class.forName()" should not load JDBC 4.0+ drivers	!
"Cloneables" should implement "clone"	!
"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used	!
"DateUtils.truncate" from Apache Commons Lang library should not be used	!
"Double.longBitsToDouble" should not be used for "int"	!
"Externalizable" classes should have no-arguments constructors	!
"HttpSecurity" URL patterns should be correctly ordered	!
"HttpServletRequest.getRequestedSessionId()" should not be used	!
"Integer.toHexString" should not be used to build hexadecimal strings	!
"InterruptedException" should not be ignored	!
"Iterator.hasNext()" should not call "Iterator.next()"	!
"Iterator.next()" methods should throw "NoSuchElementException"	!
"Lock" objects should not be "synchronized"	!
"Map.get" and value test should be replaced with single method call	!
"Object.finalize()" should remain protected (versus public) when overriding	!
"Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop	!
"Object.wait(...)" should never be called on objects that implement "java.util.concurrent.locks.Condition"	!
"Optional" should not be used for parameters	!
"Preconditions" and logging arguments should not require evaluation	!
"PreparedStatement" and "ResultSet" methods should be called with valid indices	!

"Random" objects should be reused	!
"ResultSet.isLast()" should not be used	!
"ScheduledThreadPoolExecutor" should not have 0 core threads	!
"SecureRandom" seeds should not be predictable	!
"Serializable" inner classes of non-serializable classes should be "static"	!
"StandardCharsets" constants should be preferred	!
"Stream" call chains should be simplified when possible	!
"Stream.peek()" should be used with caution	!
"Stream.toList()" method should be used instead of "collectors" when unmodifiable list needed	!
"String" calls should not go beyond their bounds	!
"String#replace" should be preferred to "String#replaceAll"	!
"StringBuilder" and "StringBuffer" should not be instantiated with a character	!
"Thread.run()" should not be called directly	!
"Thread.sleep" should not be used in tests	!
"ThreadGroup" should not be used	!
"ThreadLocal" variables should be cleaned up when no longer used	!
"ThreadLocal.withInitial" should be preferred	!
"Threads" should not be used where "Runnables" are expected	!
"URL.hashCode" and "URL.equals" should be avoided	!
"catch" clauses should do more than rethrow	!
"clone" should not be overridden	!
"close()" calls should not be redundant	!
"compareTo" results should not be checked for specific values	!
"compareTo" should not be overloaded	!
"compareTo" should not return "Integer.MIN_VALUE"	!
"default" clauses should be last	!
"else" statements should be clearly matched with an "if"	!
"enum" fields should not be publicly mutable	!
"equals" method overrides should accept "Object" parameters	!
"equals" method parameters should not be marked "@Nonnull"	!
"equals(Object obj)" and "hashCode()" should be overridden in pairs	!
"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	!
"equals(Object obj)" should test argument type	!
"finalize" should not set fields to "null"	!
"for" loop increment clauses should modify the loops' counters	!
"for" loop stop conditions should be invariant	!
"getClass" should not be used for synchronization	!
"hashCode" and "toString" should not be called on array instances	!
"indexOf" checks should not be for positive numbers	!
"iterator" should not return "this"	!
"java.nio.Files#delete" should be preferred	!
"notifyAll" should be used	!
"null" should not be used with "Optional"	!
"private" methods called only by inner classes should be moved to those classes	!
"public static" fields should be constant	!
"read" and "readLine" return values should be used	!
"read(byte[],int,int)" should be overridden	!
"readObject" should not be "synchronized"	!
"readResolve" methods should be inheritable	!
"runFinalizersOnExit" should not be called	!
"static" base class members should not be accessed via derived types	!
"static" members should be accessed statically	!
"super.finalize()" should be called at the end of "Object.finalize()" implementations	!
"switch" statements should have "default" clauses	!
"switch" statements should have at least 3 "case" clauses	!
"switch" statements should not contain non-case labels	!
"switch" statements should not have too many "case" clauses	!
"throws" declarations should not be superfluous	!
"toArray" should be passed an array of the proper type	!
"toString()" and "clone()" methods should not return null	!
"toString()" should never be called on a String object	!
"volatile" variables should not be used with compound operators	!
"wait" should not be called when multiple locks are held	!
"wait", "notify" and "notifyAll" should only be called when a lock is obviously held on an object	!
"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held	!
"write(byte[],int,int)" should be overridden	!
"writeObject" should not be the only "synchronized" code in a class	!
"List.remove()" should not be used in ascending 'for' loops	!
'serialVersionUID' field should not be set to '0L' in records	!
A "for" loop update clause should move the counter in the right direction	!
A "while" loop should be used instead of a "for" loop	!
A conditionally executed single line should be denoted by indentation	!
A field should not duplicate the name of its containing class	!
A new session should be created during user authentication	!
A secure password should be used when connecting to a database	!

Abstract class names should comply with a naming convention	⊕	⊕
Abstract classes without fields should be converted to interfaces	⊕	⊕
Abstract methods should not be redundant	⊕	⊕
All branches in a conditional structure should not have exactly the same implementation	⊕	⊖
Alternatives in regular expressions should be grouped when used with anchors	⊕	⊖
An iteration on a Collection should be performed on the type handled by the Collection	⊕	⊖
Annotation repetitions should not be wrapped	⊕	⊖
Anonymous inner classes containing only one method should become lambdas	⊕	⊖
Array designators "[]" should be located after the type in method signatures	⊕	⊖
Array designators "[]" should be on the type, not the variable	⊕	⊖
Arrays should not be copied using loops	⊕	⊖
Arrays should not be created for varargs parameters	⊕	⊖
AssertJ "assertThatThrownBy" should not be used alone	⊕	⊖
AssertJ assertions "allMatch" and "doesNotContain" should also test for emptiness	⊕	⊖
AssertJ assertions with "Consumer" arguments should contain assertion inside consumers	⊕	⊖
AssertJ configuration should be applied	⊕	⊖
AssertJ methods setting the assertion context should come before an assertion	⊕	⊖
Assertion arguments should be passed in the correct order	⊕	⊖
Assertion methods should not be used within the try block of a try-catch catching an Error	⊕	⊖
Assertions comparing incompatible types should not be made	⊕	⊖
Assertions should be complete	⊕	⊖
Assertions should not be used in production code	⊕	⊖
Assertions should not compare an object to itself	⊕	⊖
Asserts should not be used to check the parameters of a public method	⊕	⊖
Assignment of lazy-initialized members should be the last step with double-checked locking	⊕	⊖
Assignments should not be made from within sub-expressions	⊕	⊖
Assignments should not be redundant	⊕	⊖
Authorizations should be based on strong decisions	⊕	⊖
Back references in regular expressions should only refer to capturing groups that are matched before the reference	⊕	⊖
Basic authentication should not be used	⊕	⊖
Blocks should be synchronized on "private final" fields	⊕	⊖
Boolean checks should not be inverted	⊕	⊖
Boolean expressions should not be gratuitous	⊕	⊖
Boolean literals should not be redundant	⊕	⊖
Boxed "Boolean" should be avoided in boolean expressions	⊕	⊖
Boxing and unboxing should not be immediately reversed	⊕	⊖
Call to Mockito method "verify", "when" or "given" should be simplified	⊕	⊖

Constants should not be defined in interfaces	⊕	⊖
Constructors of an "abstract" class should not be declared "public"	⊕	⊖
Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes	⊕	⊖
Consumed Stream pipelines should not be reused	⊕	⊖
Cryptographic keys should be robust	⊕	⊖
Custom serialization method signatures should meet requirements	⊕	⊖
DateTimeFormatters should not use mismatched year and week numbers	⊕	⊖
Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	⊕	⊖
Deprecated annotations should include explanations	⊕	⊖
Deprecated code should be removed	⊕	⊖
Deprecated elements should have both the annotation and the Javadoc tag	⊕	⊖
Derived exceptions should not hide their parents' catch blocks	⊕	⊖
Dissimilar primitive wrappers should not be used with the ternary operator without explicit casting	⊕	⊖
Double Brace Initialization should not be used	⊕	⊖
Double-checked locking should not be used	⊕	⊖
Empty arrays and collections should be returned instead of null	⊕	⊖
Empty lines should not be tested with regex MULTILINE flag	⊕	⊖
Empty statements should be removed	⊕	⊖
Encryption algorithms should be used with secure mode and padding scheme	⊕	⊖
Enumeration should not be implemented	⊕	⊖
Equals method should be overridden in records containing array fields	⊕	⊖
Escape sequences should not be used in text blocks	⊕	⊖
Exception classes should be immutable	⊕	⊖
Exception testing via JUnit @Test annotation should be avoided	⊕	⊖
Exception testing via JUnit ExpectedException rule should not be mixed with other assertions	⊕	⊖
Exception types should not be tested using "instanceof" in catch blocks	⊕	⊖
Exceptions should be either logged or rethrown but not both	⊕	⊖
Exceptions should not be created without being thrown	⊕	⊖
Exceptions should not be thrown from servlet methods	⊕	⊖
Exceptions should not be thrown in finally blocks	⊕	⊖
Execution of the Garbage Collector should be triggered only by the JVM	⊕	⊖
Exit methods should not be called	⊕	⊖
Expressions used in "assert" should not produce side effects	⊕	⊖
Factory method injection should be used in "@Configuration" classes	⊕	⊖
Field names should comply with a naming convention	⊕	⊖

Case insensitive Unicode regular expressions should enable the "UNICODE_CASE" flag	✖
Case insensitive string comparisons should be made without intermediate upper or lower casing	✖
Catches should be combined	✖
Chained AssertJ assertions should be simplified to the corresponding dedicated assertion	✖
Character classes in regular expressions should not contain the same character twice	✖
Character classes should be preferred over reluctant quantifiers in regular expressions	✖
Child class fields should not shadow parent class fields	✖
Child class methods named for parent class methods should be overrides	✖
Cipher Block Chaining IVs should be unpredictable	✖
Cipher algorithms should be robust	✖
Class members annotated with "@VisibleForTesting" should not be accessed from production code	✖
Class names should comply with a naming convention	✖
Class names should not shadow interfaces or superclasses	✖
Class variable fields should not have public accessibility	✖
Classes extending java.lang.Thread should override the "run" method	✖
Classes from "sun.**" packages should not be used	✖
Classes named like "Exception" should extend "Exception" or a subclass	✖
Classes should not access their own subclasses during initialization	✖
Classes should not be compared by name	✖
Classes should not be empty	✖
Classes that override "clone" should be "Cloneable" and call "super.clone()"	✖
Classes with only "static" methods should not be instantiated	✖
Cognitive Complexity of methods should not be too high	✖
Collapsible "if" statements should be merged	✖
Collection constructors should not be used as java.util.function.Function	✖
Collection sizes and array length comparisons should make sense	✖
Collection.isEmpty() should be used to test for emptiness	✖
Collections should not be passed as arguments to their own methods	✖
Comma-separated labels should be used in Switch with colon case	✖
Composed "@RequestMapping" variants should be preferred	✖
Conditionally executed code should be reachable	✖
Conditionals should start on new lines	✖
Consecutive AssertJ "assertThat" statements should be chained	✖
Constant names should comply with a naming convention	✖

Fields in a "Serializable" class should either be transient or serializable	✖
Fields in non-serializable classes should not be "transient"	✖
Files opened in append mode should not be used with ObjectOutputStream	✖
Functional Interfaces should be as specialised as possible	✖
Future keywords should not be used as names	✖
Generic exceptions should never be thrown	✖
Generic wildcard types should not be used in return types	✖
Getters and setters should access the expected fields	✖
Getters and setters should be synchronized in pairs	✖
Hashes should include an unpredictable salt	✖
Identical expressions should not be used on both sides of a binary operator	✖
IllegalMonitorStateException should not be caught	✖
Inappropriate "Collection" calls should not be made	✖
Inappropriate regular expressions should not be used	✖
Inheritance tree of classes should not be too deep	✖
Inner class calls to super class methods should be unambiguous	✖
InputStream.read() implementation should not return a signed byte	✖
Insecure temporary file creation methods should not be used	✖
Instance methods should not write to "static" fields	✖
Interface names should comply with a naming convention	✖
Intermediate Stream methods should not be left unused	✖
Ints and longs should not be shifted by zero or more than their number of bits-1	✖
Invalid "Date" values should not be used	✖
JUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion	✖
JUnit assertions should not be used in "run" methods	✖
JUnit rules should be used	✖
JUnit test cases should call super methods	✖
JUnit4 @Ignored and JUnit5 @Disabled annotations should be used to disable tests and should provide a rationale	✖
JUnit5 inner test classes should be annotated with @Nested	✖
JUnit5 test classes and methods should have default package visibility	✖
JUnit5 test classes and methods should not be silently ignored	✖
JWT should be signed and verified with strong cipher algorithms	✖
Java features should be preferred to Guava	✖
Java parser failure	✖
Jump statements should not be redundant	✖

Jump statements should not occur in "finally" blocks	
LDAP connections should be authenticated	
Labels should not be used	
Lambdas containing only one statement should not nest this statement in a block	
Lambdas should be replaced with method references	
Local variable and method parameter names should comply with a naming convention	
Local variables should not be declared and then immediately returned or thrown	
Local variables should not shadow class fields	
Locks should be released	
Loggers should be named for their enclosing classes	
Loop conditions should be true at least once	
Loops should not be infinite	
Loops should not contain more than a single "break" or "continue" statement	
Loops with at most one iteration should be refactored	
Map "computeIfAbsent()" and "computeIfPresent()" should not be used to add "null" values.	
Map values should not be replaced unconditionally	
Maps with keys that are enum values should be replaced with EnumMap	
Math operands should be cast before assignment	
Members ignored during record serialization should not be used	
Method names should comply with a naming convention	
Method overrides should not change contracts	
Method parameters, caught exceptions and foreach variables' initial values should not be ignored	
Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances	
Methods and field names should not be the same or differ only by capitalization	
Methods of "Random" that return floating point values should not be used in random integer generation	
Methods returns should not be invariant	
Methods setUp() and tearDown() should be correctly annotated starting with JUnit4	
Methods should not be empty	
Methods should not be named "toString", "hashCode" or "equals"	
Methods should not be too complex	
Methods should not call same-class methods with incompatible "@Transactional" values	
Methods should not have identical implementations	
Methods should not have too many parameters	
Methods should not have too many return statements	
Methods should not return constants	

Min and max used in combination should not always return the same value	✖️
Mobile database encryption keys should not be disclosed	✖️
Mocking all non-private methods of a class should be avoided	✖️
Modifiers should be declared in the correct order	✖️
Multiline blocks should be enclosed in curly braces	✖️
Multiple variables should not be declared on the same line	✖️
Mutable fields should not be "public static"	✖️
Names of regular expressions named groups should be used	✖️
Neither "Math.abs" nor negation should be used on numbers that could be "MIN_VALUE"	✖️
Nested "enum"s should not be declared static	✖️
Nested blocks of code should not be left empty	✖️
Nested code blocks should not be used	✖️
Non-constructor methods should not have the same name as the enclosing class	✖️
Non-primitive fields should not be "volatile"	✖️
Non-public methods should not be "@Transactional"	✖️
Non-serializable classes should not be written	✖️
Non-serializable objects should not be stored in "HttpSession" objects	✖️
Non-thread-safe fields should not be static	✖️
Null checks should not be used with "instanceof"	✖️
Null pointers should not be dereferenced	✖️
Null should not be returned from a "Boolean" method	✖️
Nullness of parameters should be guaranteed	✖️
Objects should not be created only to "getClass"	✖️
Only one method invocation is expected when testing checked exceptions	✖️
Only one method invocation is expected when testing runtime exceptions	✖️
Only static class initializers should be used	✖️
OpenSAML2 should be configured to prevent authentication bypass	✖️
Operator "instanceof" should be used instead of "A.class.isInstance()"	✖️
Optional value should only be accessed after calling isPresent()	✖️
Overrides should match their parent class methods in synchronization	✖️
Overriding methods should do more than simply call the same method in the super class	✖️
Package declaration should match source file directory	✖️
Package names should comply with a naming convention	✖️
Packages containing only "package-info.java" should be removed	✖️
Parameters should be passed in the correct order	✖️
Parentheses should be removed from a single lambda input parameter when its type is inferred	✖️
Parsing should be used to convert "Strings" to primitives	✖️
Passwords should not be stored in plain-text or with a fast hashing algorithm	✖️
Pattern Matching for "instanceof" operator should be used instead of simple "instanceof" + cast	✖️
Permitted types of a sealed class should be omitted if they are declared in the same file	✖️
Persistent entities should not be used as arguments of "@RequestMapping" methods	✖️
Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls	✖️
Primitives should not be boxed just for "String" conversion	✖️
Printf-style format strings should be used correctly	✖️
Printf-style format strings should not lead to unexpected behavior at runtime	✖️
Private fields only used as local variables in methods should become local variables	✖️
Public constants and fields initialized at declaration should be "static final" rather than merely "final"	✖️
Raw byte values should not be used in bitwise operations in combination with shifts	✖️
Raw types should not be used	✖️
Records should be used instead of ordinary classes when representing immutable data structure	✖️
Redundant casts should not be used	✖️
Redundant constructors/methods should be avoided in records	✖️
Redundant pairs of parentheses should be removed	✖️
Reflection should not be used to check non-runtime annotations	✖️
Reflection should not be used to increase accessibility of classes, methods, or fields	✖️
Reflection should not be used to increase accessibility of records' fields	✖️
Regex alternatives should not be redundant	✖️
Regex boundaries should not be used in a way that can never be matched	✖️
Regex lookahead assertions should not be contradictory	✖️
Regex patterns following a possessive quantifier should not always fail	✖️
Regexes containing characters subject to normalization should use the CANON_EQ flag	✖️
Regular expressions should be syntactically valid	✖️
Regular expressions should not be too complicated	✖️
Regular expressions should not overflow the stack	✖️
Related "if/else if" statements should not have the same condition	✖️
Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string	✖️
Repeated patterns in regular expressions should not match the empty string	✖️
Resources should be closed	✖️
Restricted Identifiers should not be used as Identifiers	✖️
Return of boolean expressions should not be wrapped into an "if-then-else" statement	✖️

Return values from functions without side effects should not be ignored	
Return values should not be ignored when they contain the operation status code	
Sections of code should not be commented out	
Server certificates should be verified during SSL/TLS connections	
Server hostnames should be verified during SSL/TLS connections	
Servlets should not have mutable instance fields	
Short-circuit logic should be used in boolean contexts	
Silly String operations should not be made	
Silly bit operations should not be performed	
Silly equality checks should not be made	
Silly math should not be performed	
Similar tests should be grouped in a single Parameterized test	
Simple string literal should be used for single line strings	
Single-character alternations in regular expressions should be replaced with character classes	
Standard outputs should not be used directly to log anything	
Static fields should not be updated in constructors	
Static non-final field names should comply with a naming convention	
String literals should not be duplicated	
String multiline concatenation should be replaced with Text Blocks	
String offset-based methods should be preferred for finding substrings from offsets	
String.valueOf() should not be appended to a String	
Strings and Boxed types should be compared using "equals()"	
Strings should not be concatenated using '+' in a loop	
Subclasses that add fields should override "equals"	
Switch arrow labels should not use redundant keywords	
Switch cases should end with an unconditional "break" statement	
Synchronization should not be done on instances of value-based classes	
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	
Ternary operators should not be nested	
Test classes should comply with a naming convention	
Test methods should not contain too many assertions	
TestCases should contain tests	
Tests method should not be annotated with competing annotations	
Tests should be stable	
Tests should include assertions	

Text blocks should not be used in complex expressions	⊕ ✓
The Object.finalize() method should not be called	✗ ↗
The Object.finalize() method should not be overridden	✗ ↗
The default unnamed package should not be used	✗ ↗
The diamond operator ("<>") should be used	✗ ↗
The non-serializable super class of a "Serializable" class should have a no-argument constructor	✗ ↗
The regex escape sequence \cX should only be used with characters in the @- _ range	✗ ↗
The signature of "finalize()" should match that of "Object.finalize()"	✗ ↗
The value returned from a stream read should be checked	✗ ↗
Throwable and Error should not be caught	✗ ↗
Track uses of "FIXME" tags	✗ ↗
Track uses of "TODO" tags	✗ ↗
Try-catch blocks should not be nested	✗ ↗
Try-with-resources should be used	✗ ↗
Two branches in a conditional structure should not have exactly the same implementation	✗ ↗
Type parameter names should comply with a naming convention	✗ ↗
Type parameters should not shadow other type parameters	✗ ↗
URIs should not be hardcoded	✗ ↗
Unary prefix operators should not be repeated	✗ ↗
Unicode Grapheme Clusters should be avoided inside regex character classes	✗ ↗
Unnecessary imports should be removed	✗ ↗
Unused "private" classes should be removed	✗ ↗
Unused "private" fields should be removed	✗ ↗
Unused "private" methods should be removed	✗ ↗
Unused assignments should be removed	✗ ↗
Unused labels should be removed	✗ ↗
Unused local variables should be removed	✗ ↗
Unused method parameters should be removed	✗ ↗
Unused type parameters should be removed	✗ ↗
Utility classes should not have public constructors	✗ ↗
Value-based classes should not be used for locking	✗ ↗
Values should not be uselessly incremented	✗ ↗
Vararg method arguments should not be confusing	✗ ↗
Variables should not be self-assigned	✗ ↗
Weak SSL/TLS protocols should not be used	✗ ↗

Week Year ("YYYY") should not be used for date formatting	✗ ↗
Whitespace and control characters in literals should be explicit	✗ ↗
Whitespace for text block indent should be consistent	✗ ↗
XML parsers should not allow inclusion of arbitrary files	✗ !
XML parsers should not be vulnerable to Denial of Service attacks	✗ !
XML parsers should not be vulnerable to XXE attacks	✗ !
XML parsers should not load external schemas	✗ !
XML signatures should be validated securely	✗ !
Zero should not be a possible denominator	✗ ↗

Result of SonarLint

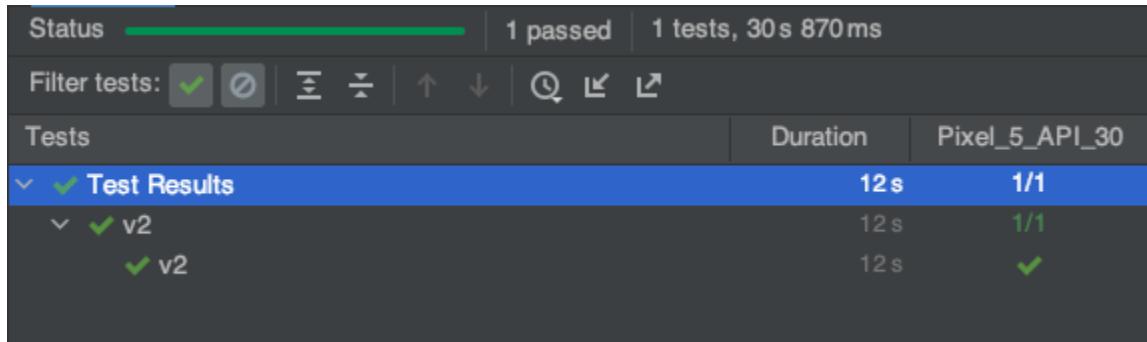
This is the result of running SonarLint rules throughout whole project:

The screenshot shows the Android Studio interface with the SonarLint plugin active. The main code editor displays a Java file named LevelOne.java. The bottom right corner of the code editor shows a red warning icon, indicating there are issues. The SonarLint tool window is open, titled 'Resource Manager'. It lists 'Found 8 issues in 8 files' across several Java files: LevelEight.java, LevelFive.java, LevelFour.java, LevelOne.java, LevelSeven.java, LevelSix.java, LevelThree.java, and LevelTwo.java. Each file has one issue related to the use of the 'Handler' class, which is marked as deprecated. The 'Rule' tab is selected in the SonarLint window. The bottom status bar of the IDE shows various tool icons and the message 'Analysis of all project files done few seconds ago'.

Each of the different level classes 1-8, use a handler to allow for there to be a runnable object in the code. This runnable object is the timer and its function is to run continuously keeping track of the total time playing the game. SonarLint says it should not be used because it is deprecated. This is not correct for our intended use so the handler must be used in order for the timer to function properly. For this reason and because it's only a minor bug it was decided that this wasn't a "true" issue and can be ignored.

Code Coverage

To make sure the application runs correctly we used Espresso, a tool known to write concise, automated and reliable Android UI tests. We used espresso to test the Main screen, sub menu screen, and the instructions screen to make sure that each screen displayed their proper elements each and every time the app is run. These are the test files used to insure each screen was working properly:



```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class v2 {

    @Rule
    public ActivityTestRule<NavigationActivity> mActivityTestRule = new ActivityTestRule<>(NavigationActivity.class);

    @Test
    public void v2() {
        ViewInteraction textView = onView(
            allOf(withId(R.id.Play_button_text), withText("Play Game"),
                withParent(allOf(withId(R.id.main_menu_layout),
                    withParent(withId(R.id.nav_host_fragment)))),
                isDisplayed()));
        textView.check(matches(isDisplayed()));

        ViewInteraction textView2 = onView(
            allOf(withId(R.id.Play_button_text), withText("Play Game"),
                withParent(allOf(withId(R.id.main_menu_layout),
                    withParent(withId(R.id.nav_host_fragment)))),
                isDisplayed()));
        textView2.check(matches(isDisplayed()));

        ViewInteraction appCompatImageView = onView(
            allOf(withId(R.id.play_game_button), withContentDescription("sets screen back to submenu"),
                childAtPosition(
                    allOf(withId(R.id.main_menu_layout),
                        childAtPosition(
                           (withId(R.id.nav_host_fragment),
                                position: 0),
                            position: 8)));
    }
}
```

```
        isDisplayed());
appCompatImageView.perform(click());

ViewInteraction textView3 = onView(
    allOf(withId(android.R.id.text1), withText("Traditional"),
          withParent(allOf(withId(R.id.game_mode_spinner), withContentDescription( text: "CHOOSE MODE"),
                      withParent(withId(R.id.frameLayout)))),
          isDisplayed()));
textView3.check(matches(isDisplayed()));

ViewInteraction button = onView(
    allOf(withId(R.id.level_1_button), withText("1"),
          withParent(allOf(withId(R.id.frameLayout),
                      withParent(withId(R.id.nav_host_fragment)))),
          isDisplayed()));
button.check(matches(isDisplayed()));

ViewInteraction button2 = onView(
    allOf(withId(R.id.level_2_button), withText("2"),
          withParent(allOf(withId(R.id.frameLayout),
                      withParent(withId(R.id.nav_host_fragment)))),
          isDisplayed()));
button2.check(matches(isDisplayed()));

ViewInteraction button3 = onView(
    allOf(withId(R.id.level_3_button), withText("3"),
          withParent(allOf(withId(R.id.frameLayout),
                      withParent(withId(R.id.nav_host_fragment)))),
          isDisplayed()));
button3.check(matches(isDisplayed()));

ViewInteraction button4 = onView(
    allOf(withId(R.id.level_4_button), withText("4"),
          withParent(allOf(withId(R.id.frameLayout),
                      withParent(withId(R.id.nav_host_fragment)))),
          isDisplayed()));
```

```
button4.check(matches(isDisplayed()));

ViewInteraction button5 = onView(
    allOf(withId(R.id.level_5_button), withText("5"),
        withParent(allOf(withId(R.id.frameLayout),
            withParent(withId(R.id.nav_host_fragment))),
        isDisplayed())));
button5.check(matches(isDisplayed()));

ViewInteraction button6 = onView(
    allOf(withId(R.id.level_6_button), withText("6"),
        withParent(allOf(withId(R.id.frameLayout),
            withParent(withId(R.id.nav_host_fragment))),
        isDisplayed())));
button6.check(matches(isDisplayed()));

ViewInteraction button7 = onView(
    allOf(withId(R.id.level_7_button), withText("7"),
        withParent(allOf(withId(R.id.frameLayout),
            withParent(withId(R.id.nav_host_fragment))),
        isDisplayed())));
button7.check(matches(isDisplayed()));

ViewInteraction button8 = onView(
    allOf(withId(R.id.level_8_button), withText("8"),
        withParent(allOf(withId(R.id.frameLayout),
            withParent(withId(R.id.nav_host_fragment))),
        isDisplayed())));
button8.check(matches(isDisplayed()));

ViewInteraction button9 = onView(
    allOf(withId(R.id.submenu_back_button), withText("MAIN MENU"),
        withParent(allOf(withId(R.id.frameLayout),
            withParent(withId(R.id.nav_host_fragment))),
        isDisplayed())));
button9.check(matches(isDisplayed()));
```

```
ViewInteraction button11 = onView(
    allOf(withId(R.id.instructions_button), withText("INSTRUCTIONS"),
        withParent(allOf(withId(R.id.frameLayout),
            withParent(withId(R.id.nav_host_fragment)))),
        isDisplayed()));
button11.check(matches(isDisplayed()));

ViewInteraction materialButton = onView(
    allOf(withId(R.id.instructions_button), withText("Instructions"),
        childAtPosition(
            allOf(withId(R.id.frameLayout),
                childAtPosition(
                   (withId(R.id.nav_host_fragment),
                        position: 0)),
                position: 7)),
        isDisplayed()));
materialButton.perform(click());

ViewInteraction textView4 = onView(
    allOf(withId(R.id.back_instruction_button_text), withText("Back"),
        withParent(allOf(withId(R.id.frameLayout2),
            withParent(withId(R.id.nav_host_fragment)))),
        isDisplayed()));
textView4.check(matches(isDisplayed()));

ViewInteraction appCompatImageView2 = onView(
    allOf(withId(R.id.back_instruction_button), withContentDescription("sets screen back to submenu"),
        childAtPosition(
            allOf(withId(R.id.frameLayout2),
                childAtPosition(
                   (withId(R.id.nav_host_fragment),
                        position: 0)),
                position: 4)),
        isDisplayed()));
appCompatImageView2.perform(click());
```

```

        ViewInteraction appCompatSpinner = onView(
            allOf(withId(R.id.game_mode_spinner), withContentDescription("CHOOSE MODE"),
                childAtPosition(
                    allOf(withId(R.id.frameLayout),
                        childAtPosition(
                           (withId(R.id.nav_host_fragment),
                                position: 0)),
                        position: 5),
                isDisplayed())));
        appCompatSpinner.perform(click());

        ViewInteraction checkedTextView = onView(
            allOf(withId(android.R.id.text1), withText("Time Trial"),
                withParent(withParent(IsInstanceOf.<~>instanceOf(android.widget.FrameLayout.class))),
                isDisplayed()));
        checkedTextView.check(matches(isDisplayed()));

        DataInteraction appCompatCheckedTextView = onData(anything())
            .inAdapterView(childAtPosition(
                withClassName(is("android.widget.PopupWindow$PopupBackgroundView")),
                position: 0))
            .atPosition(1);
        appCompatCheckedTextView.perform(click());
    }
}

```

```

private static Matcher<View> childAtPosition(
    final Matcher<View> parentMatcher, final int position) {

    return new TypeSafeMatcher<View>() {
        @Override
        public void describeTo(Description description) {
            description.appendText("Child at position " + position + " in parent ");
            parentMatcher.describeTo(description);
        }

        @Override
        public boolean matchesSafely(View view) {
            ViewParent parent = view.getParent();
            return parent instanceof ViewGroup && parentMatcher.matches(parent)
                && view.equals(((ViewGroup) parent).getChildAt(position));
        }
    };
}

```

The tests above ensured the appearance of buttons that are essential to app operation.

- On the Main Menu screen we made sure that the ‘Play Game’ button appeared. This is the entry to the game that allows the actor to engage with our app.
- When we moved to the Sub Menu screen, we had to be sure that the drop down menu that allowed the actor to select what mode they wanted to play appeared. This is crucial for the user to decide which game mode they would like to enter. In our latest version, we also added 7 new levels. All 8 buttons are seen on this Sub Menu screen, however

only the first is selectable. We did want to ensure that all buttons appeared. Also, we needed to be sure that the ‘Instructions’ and the ‘Main Menu’ buttons were visible.

- The Main Menu button allowed the user to do just that, go back to the Main Menu.
- Lastly, We had to make sure that the user had access to the rules of the game, as well as the rules of engagement with the app. This is why we made a test to ensure that the ‘Instructions’ button appeared.

The goal for a memory card game is to create a board of randomized cards. Because of the randomness of each object on the screen there was no way of running automated testing through the objects on the game board. To test these screens and some sort of code coverage over them we manually did tests by hand making sure each of the elements on each screen functions properly. By doing dozens of tests each we gained confidence that this part of the code is “covered” and working.

Responsibilities & Reflections

Matt Ferretti

Responsibilities:

- Scrum Master
- Coding the game logic of the app
- Creating the look of multiple GUI screens
- Implementing and making code comply with SonarLink rules
- Documentation Editor

Reflection:

In the beginning we were a little too ambitious on what we could accomplish but we realized this early enough and revised the project's specifications. Also, in the beginning it was difficult to get the basic functionality to work as this was the first time I ever worked on a mobile application before. Because of this I had to figure out and learn a lot of things myself. However, I enjoyed figuring out and learning these new things as it gave me better insight into how mobile applications work and run. As a computer science student I plan on going into software so I really liked understanding how software runs on mobile devices. Also, It was very rewarding and fun to just play the game and see your program and solutions work.

There were a lot of issues that came up during the development process but as a team we were able to work together and figure out a solution. In the end we accomplished our goal and created a full functioning memory card game. And I'm happy we created a memory card game. I really like card games and I think that is part of the reason it was really fun learning to build one on an Android device. Because of this project I plan on taking a mobile application development class as this has peaked my interest.

Steve Meadows

Responsibilities:

- Randomization algorithms for game logic
- Writing test units
- Generating UML diagrams
- Collaborator on Use Case Diagram
- Writing Use Case Descriptions
- Implementing use of Scrum board
- Documentation Contributor
- Implemented Time Trial Feature

Reflection:

Experience is the best teacher. This was one of those moments for me where I was able to see the tools and applications we have been studying come together. This was certainly a challenge and will continue to be, but I'm sure that we can keep creating ideas and press the project forward.

It seemed early on that we were all headed in a direction where we wanted to create some sort of card game. We floated a few ideas but finally landed on Memory. This was a game we were all familiar with and that we felt comfortable pulling out of abstraction and putting into more of a concrete form.

I was initially thinking more toward a desktop game, but one of our team members felt comfortable using Android Studio, and we all felt it might offer a bit of versatility to add some more features. We soon figured out that Android Studio was a vast program and dialed back on some of our loftier ambitions. I have very limited experience working with GUI based programming, so for me, there was a steep learning curve and I'd hoped to glean as much experience as possible to be able to utilize some take-aways and learn from my group partners. I was, however, able to use my algorithm writing skills to help adapt the code to build a fully random card deck for the game.

The second release seemed to be much easier for implementation. Our experience with the code was better so I felt more comfortable navigating the structure of the program.

All-in-all, this was a great learning experience in building something from scratch, using tools to plan and manage a project, and gaining experience using tools to test and standardize code.

Isabel Albaitis

Responsibilities:

- UI design and implementation
- Creating use case diagrams
- Handling screen transitions
- Implementation of passing data between screens
- Solving Android-specific issues

When we first started this project, I knew I wanted to create a mobile app. At first I wanted to create an iOS app, but since we all were familiar with Java. It had seemed that my whole group thought it would be the easiest for all of us to implement given our skills. I was the only one, however, who had any experience building an Android app, so I took the lead on solving Android-specific issues as they arose.

My first main responsibility was designing the GUI of the game board and how to pass the data generated by the user's interactions to the different screens. For me, a major learning curve was learning how the specific architecture of our app affected different diagrams and testing.

I think that the members of this group complemented each other's gaps in knowledge. Personally, I had never created a game before, so this was a major learning experience for me, so I was able to learn from them in those aspects. I generally enjoyed taking the lead on Android-specific responsibilities such as the UI implementation, handling, and validating user-generated data. I think that this was also a good experience for me to be able to learn how to follow a process of developing a program, and how to automate the process using external tools.

I liked that I was able to have control of the way our app looked, and making sure that our UI was as intuitive as possible. I feel that our team made great improvements from release one to release two in this aspect. I also think that we as a group improved our communications with one another throughout the course of the semester, but especially after our first release. We agreed that if we were to do this over again, we would have made one base level class and extend it to create each level.

I think overall this was a great learning experience for me, as I was not only able to work on my technical skills but also my communication and teamwork skills.