Технически университет - София

Проект

Лов на музеи

Програмиране за мобилни устройства

Автор:
Богомил Коларов (№121215151)
Курс 3, Група 51

Ръководител:
доц. А. Ташева

май 2019

Съдържание

Увод

Основният проблем, пред който е поставен този проект, е създаването на игра с образователна цел, чрез която играчът да се запознае с някои от българските музеи в София.

За реализацията трябва да се вземе предвид следното:

- Потребителят трябва да е над 13 години (достатъчно голям, за да се справя с придвижването в града)
- Лесно навигиране
- Нужда от актуално следене на местоположението на играча, когато етапът на играта го изисква
- Опростен изглед
- Интерактивен интерфейс за запомнянето на думите
- Загадка, която да свърже с думите
- Запазване на прогреса на играта

С Приложението *“Лов на музеи”* - разработено на Unity3d и Android платформа, се цели да се ангажира вниманието на потребителя, насочвайки го към определени думи, които той после да свърже с даден музей, до чието местоположение трябва да стигне. Комбинацията от задачите за изпълнение има идеята да създаде някакво изживяване у играча, карайки го да запомни информацията.

Програмата е снабдена с различни изгледи, които да водят потребителя от познаването на думи, към отгатването на загадка, до намиране местоположението на даден музей:

- Екран за начало на игра
- Екран за избор измежду вече съществуващи игри
- Екран - интерактивна игра
- Екран със загадка и възможни отговори към нея
- Екран с местоположението на даден музей
- Екран за край на играта

Избор за създаване на нова игра или продължаване на вече съществуваща такава е представен в началния екран. Нова игра може да бъде създадена и от екрана, визуализиращ вече съществуващите игри.

Една игра е разделена на нива, всяко от което има 3 етапа:

- Откриване на думи
- Отгатване на загадка
- Намиране на местоположение

Преминаването между етапите е линейно и се случва в реда, изброен по-горе. Ако потребителят е прекъснал своята игра, той ще може да я продължи от нивото и етапа, до които последно е стигнал.

Изисквания

Функционални изисквания

Номер	Изискване	Приоритет
0	**База данни**	Задължително
0.1	Съхраняване на игра	Задължително
0.2	Запазване на прогрес	Задължително
0.3	Съхраняване на множество игри	Задължително
1	**Интерактивна игра**	Задължително
1.1	Динамично генериране на нивото на играта	Висок
2	**Загадка**	Задължително
2.1	Динамично зареждане на загадката	Висок
2.2	Динамично зареждане на отговорите	Висок
2.3	Показване на диалог с откритите думи	Висок
3	**Местоположение**	Задължително
3.1	Карта, показваща местоположението на музея	Задължително
3.2	Проследяване на местоположението в реално време	Висок
3.3	Препращане на потребителя към Google Maps	Среден
4	**Управление на игрите**	Задължително
4.1	Създаване на нова игра	Задължително

4.2	Съхранение на множество игри	Висок
4.3	Задаване на име на играта от потребителя	Висок
5	**Навигация**	Задължително
5.1	От начален екран към новосъздадената игра	Задължително
5.2	От интерактивния екран към екран със загадка	Задължително
5.3	От екрана със загадка към екрана за местоположение	Задължително
5.4	От начален екран към екрана със загадка (при продължаване на съществуваща игра)	Висок
5.5	От начален екран към екрана за местоположение (при продължаване на съществуваща игра)	Висок
5.6	От екрана за местоположение към екран за край на играта	Задължително
5.7	Обратна навигация	Задължително

Нефункционални изисквания

Номер	Изисквания	Приоритет
0	Android 5.0 (API level 23)	Задължително
1	Поддръжка на размери на дисплея между 4.5” и 7”	Задължително
2	Използване на многонишковост за зареждане на данните	Задължително

Потребителски истории

Номер	Като	Аз искам да	Така че
0	Играч	Създавам игра	Да мога да я играя
1	Играч	Избирам букви	Да мога да позная дума
2	Играч	Мога да видя намерените думи	Мога да отгатна загадката
3	Играч	Видя местоположението на даден музей	Мога да го намеря
4	Играч	Ми се следи местоположението в реално време	Автоматично да бъда препращан към следващия етап от играта
5	Играч	Ми се пази прогресът на играта	Да не трябва да започвам от начало
6	Играч	Мога да избирам измежду различните започнати от мен игри	Да не съм принуден винаги да започвам нова игра и да не съм принуден да продължавам само определена започната игра
7	Играч	Мога да бъда препратен към Google Maps	Получа напътствия как да стигна до даден музей
8	Играч	Да бъда	Да предприема

		известен, когато местоположение то не може да бъде засечено	мерки за разрешаването му

Критерии за приемане

Номер	**Критерий за приемане**
0	Актуално показване на текущото местоположение
1	Автоматично продължаване на играта при влизане в радиус от 150 метра от местоположението на музей
2	Хронологично представяне на игрите
3	Представяне на игрите в списък
4	Избиране на игра от списък
5	Известяване чрез диалог при липса на местоположение
6	База данни, в която се пазят започнатите игри и техният прогрес

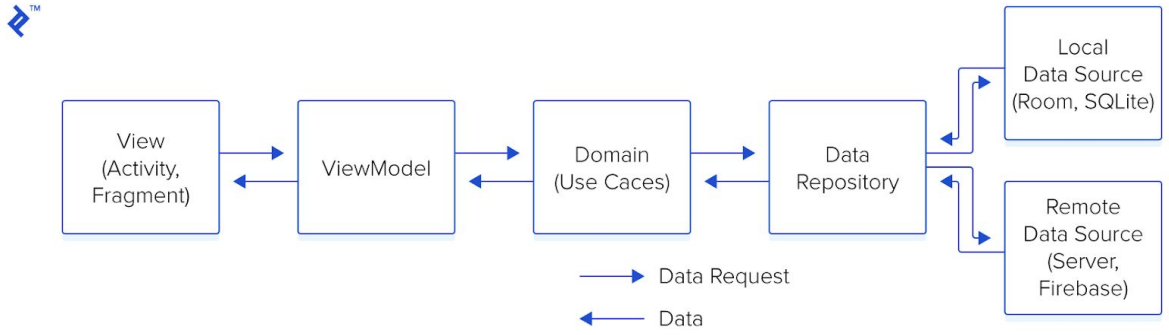
Проектиране

Приложението не изисква от потребителя голяма вариация от входни данни. Това позволява създаването на по-прост и удобен за ползване потребителски интерфейс. Използвани са две теми, които целят потребителят ясно да разбира в коя част от приложението се намира. Темите са *“извън игра”* и “*в игра”*.

Взета е предвид високата консумация на батерия при взимането на местоположението в реално време, затова то се следи само докато потребителят е на този етап от играта и приложението е активно.

Заради сложността на екрана с кръстословицата, за неговата разработка е избрана платформата за игри Unity3d. Тя улеснява създаването на игри и ги дистрибутира към различни платформи. Заради използването на тази платформа, проектът е разделен на 2 модула - unity module и native android module.

Архитектурата на native android проекта трябва да изисква разделянето му на отделни, възможно най-независими един от друг компоненти. Това ще позволи тяхното лесно заменяне и удобство при тестване (JUnit). За целта е избрана MVVM (Model View ViewModel) архитектурата в комбинация с чиста архитектура (Clean architecture - фиг. 1).



Фиг. 1

За локално съхранение на данните се използва SQLite база данни, управлявана чрез Android Room. Фреймуъркът позволява лесна работа с базата и осигурява типова сигурност и валидация на заявките по време на компилация.

За улеснение на зависимостите от други класове на отделните компоненти на приложението се използва библиотеката за *dependency injection* Dagger 2, активно разработвана от Google.

С цел намаляване на товара на основната нишка е използвано така нареченото реактивно програмиране (reactive programming), посредством RxJava 2. С негова помощ достъпът до базата и обработването на данни лесно се извършва асинхронно на отделни нишки, без да се причинява лошо потребителско изживяване (UX).

За защита от нежелано поведение на приложението при случаи, зависими от цикъла на екраните (Activity/Fragment Lifecycle), View компонентът от архитектурата се актуализира чрез data binding и LiveData. По този начин няма риск от достъпване на някой излгед, който вече е унищожен.

Диаграма на потока в приложението

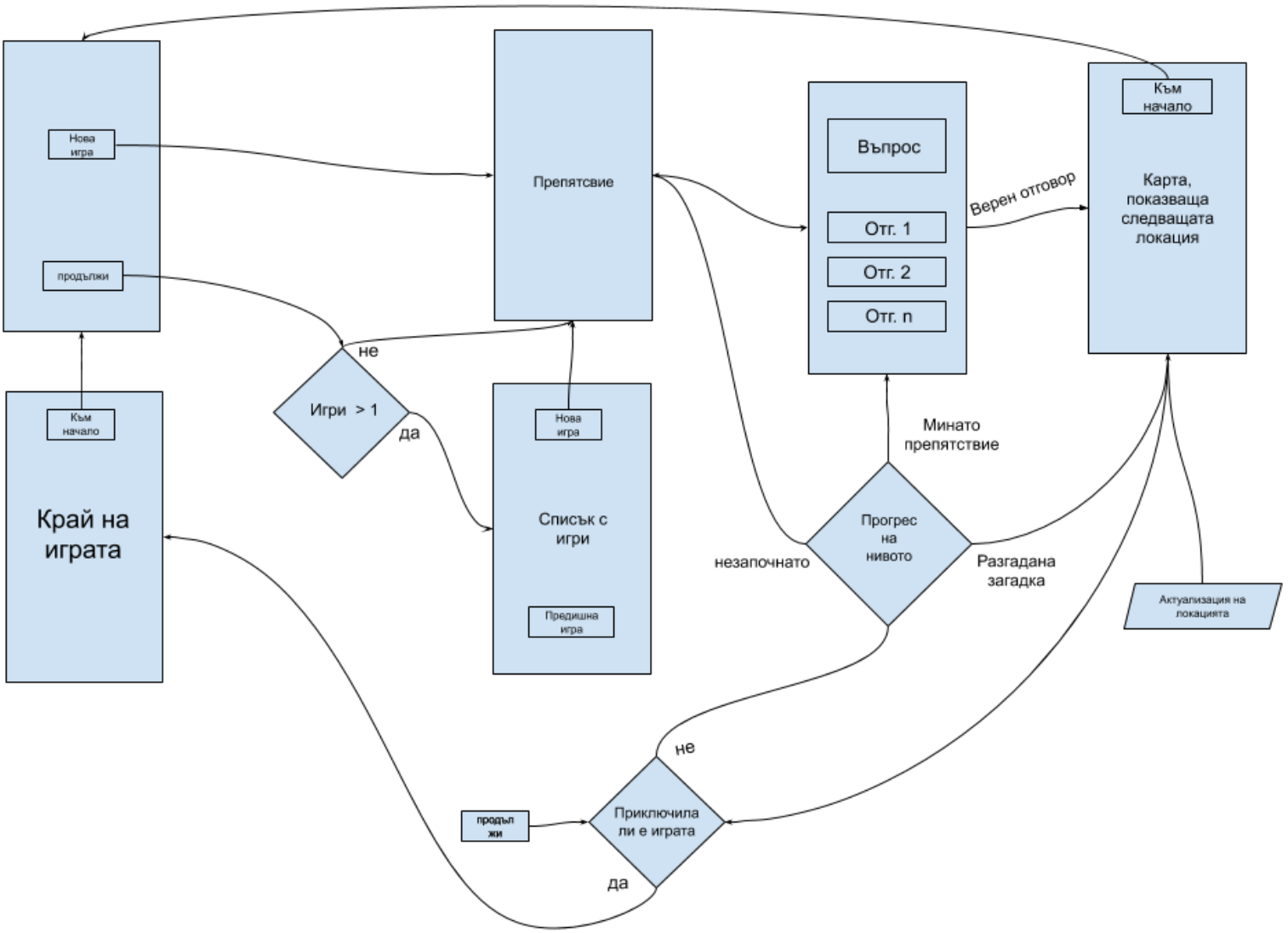
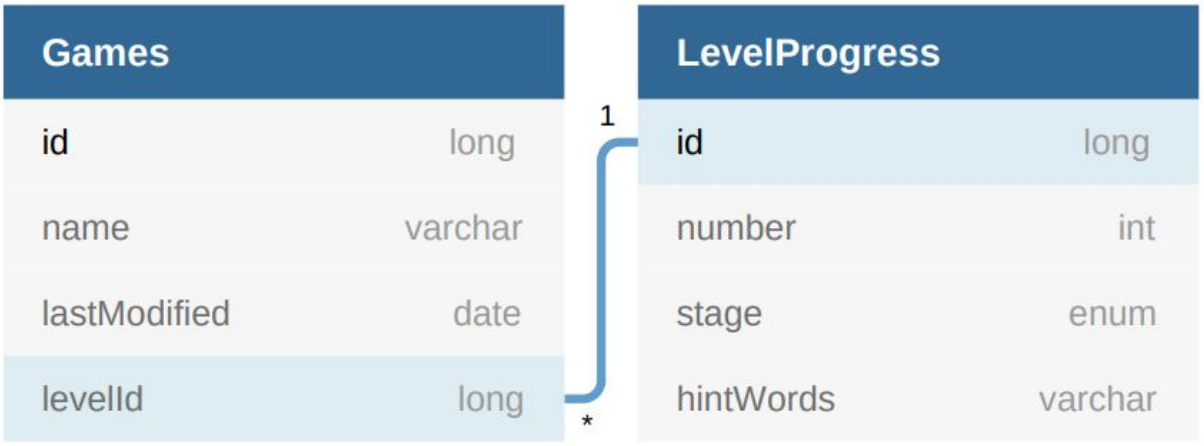


Схема на базата данни



Анализ на други разработки

При започването на даден проект е добре да се проверят вече съществуващите му алтернативи, ако има такива. Трябва да се изследват предимствата и недостатъците на готовите решения, целейки да се създаде по-добър продукт.

Алтернативи на *Лов на музеи* не бяха намерени. Все пак проектът е вдъхновен от следните неща:

- Игра за търсене из града, организирана от т. нар. “стаи за бягство” на Енигмания и Room 66. Това не са софтуерни приложения, а игри в реалния свят, които се състоят в това да се отиде от едно място на друго, решавайки загадки и преодолявайки препятствия.
- Мобилна игра Pokemon GO от Niantic Inc. - Това е игра, базирана на местоположение, в която едно от основните неща е да се събират покемони. За целта играчът трябва да обикаля и да ги търси.

- Triviador - мобилна и уеб игра за любознателните. В нея се отговаря на въпроси от различен аспект, включително и на българска тематика. Целта е да се завладяват територии и превземат крепости.

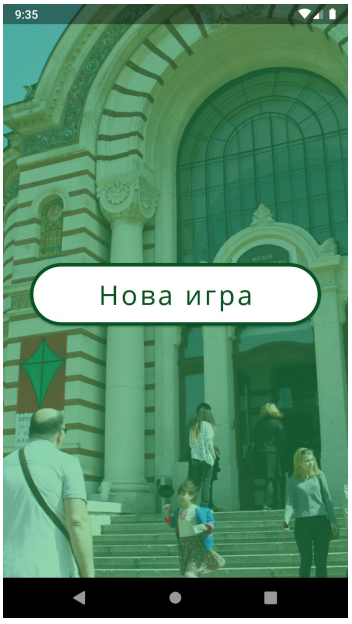
Потребителско ръководство

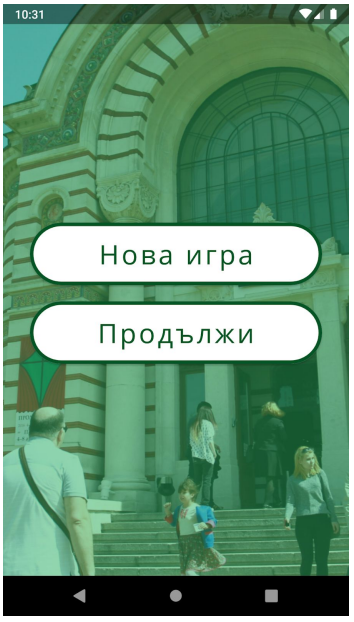
Начален екран

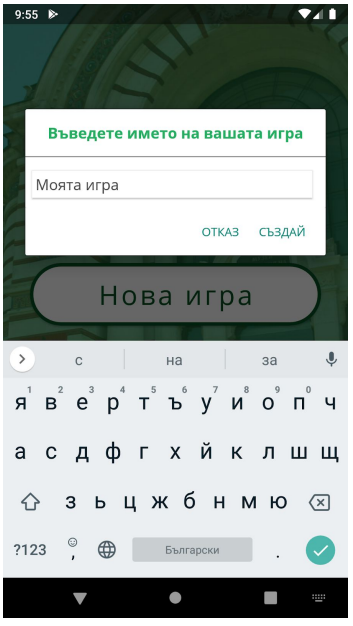
Това е стандартният екран при стартиране на приложението. Предлага опциите за създаване на игра и продължаване на вече съществуваща, ако има такава.

Съдържа:

1. Бутон за създаване на нова игра
2. При натискане на бутона за създаване на игра се показва диалог, изискващ име за играта.
3. Бутон за продължаване на съществуваща игра. При една единствена съществуваща игра, приложението ще я стартира автоматично. В противен случай ще се покажа екранът за избор на игра.
4. Диалог за създаване на нова игра

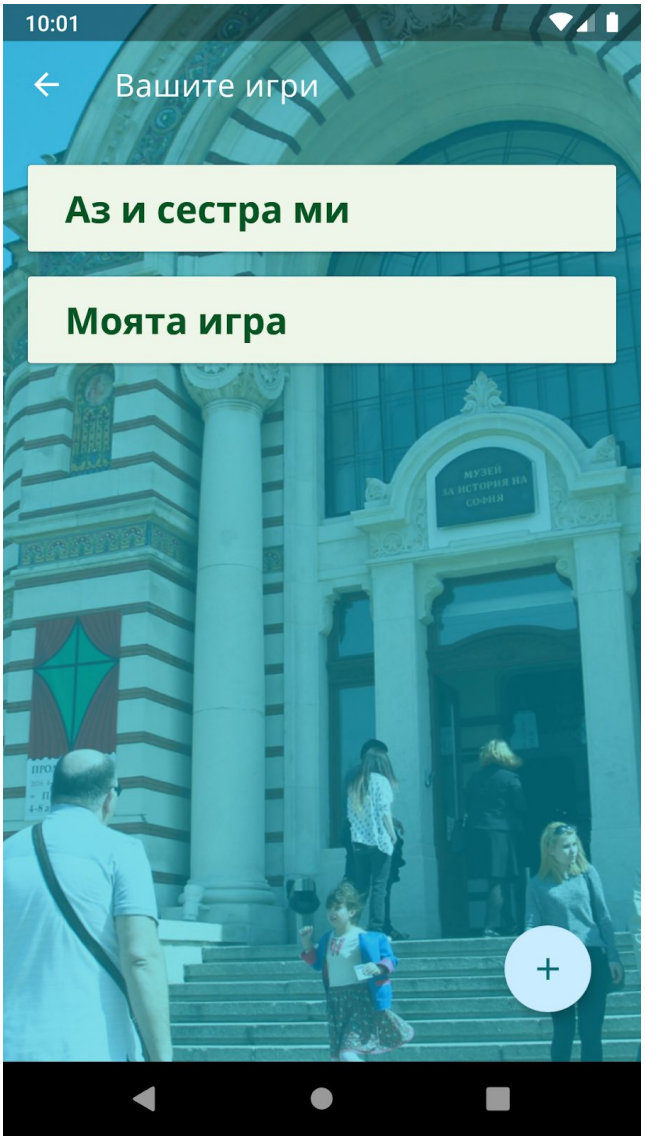






Екран за избор на игра

Чрез този екран потребителят може да види започнатите му игри и да избере коя от тях да продължи.



Съдържа:

1. Списък със съществуващи игри
2. Бутон за създаване на нова игра
3. При натискане на бутона за създаване на игра се показва диалог, изискващ име за играта.
4. Диалог за създаване на нова игра
5. При избиране на игра, тя започва

Екран за отгатване на думи

Този екран показва подобен на кръстословица изглед. Играчът трябва да избере буквите в правилната последователност, за да открие дума. При намерена дума, буквите изчезват. Целта е анимираният човек да премине от другата страна. При избор на грешна последователност, блоковете се оцветяват в червено. При правилна се оцветяват в зелено.

Съдържа:

1. Бутон за връщане назад
2. Бутон за помощ (показва диалог, който упътва потребителя, че трябва да реши кръстословицата)
3. Поле за избраните букви
4. При успешно преминаване, играта продължава напред

Екран за разгадаване на загадка

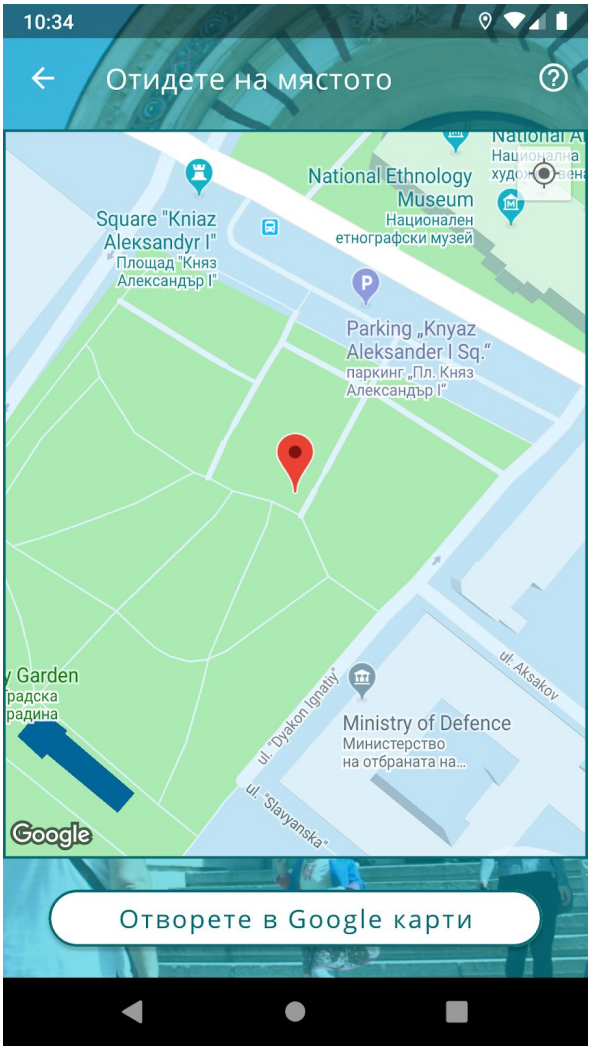
На този екран потребителят трябва да отговори правилно на зададения въпрос, за да продължи. Думите от предходния екран са подсказка за отговора. Може да ги види чрез бутона горе в дясно.

Съдържа:

1. Въпрос
2. Възможни отговори
3. Бутон за помощ, който показва думите-подсказки
4. Бутон за връщане назад
5. При правилен отговор играта продължава напред.

Екран за намиране на местоположение

Този екран показва местоположението на музея, до който той трябва да стигне.

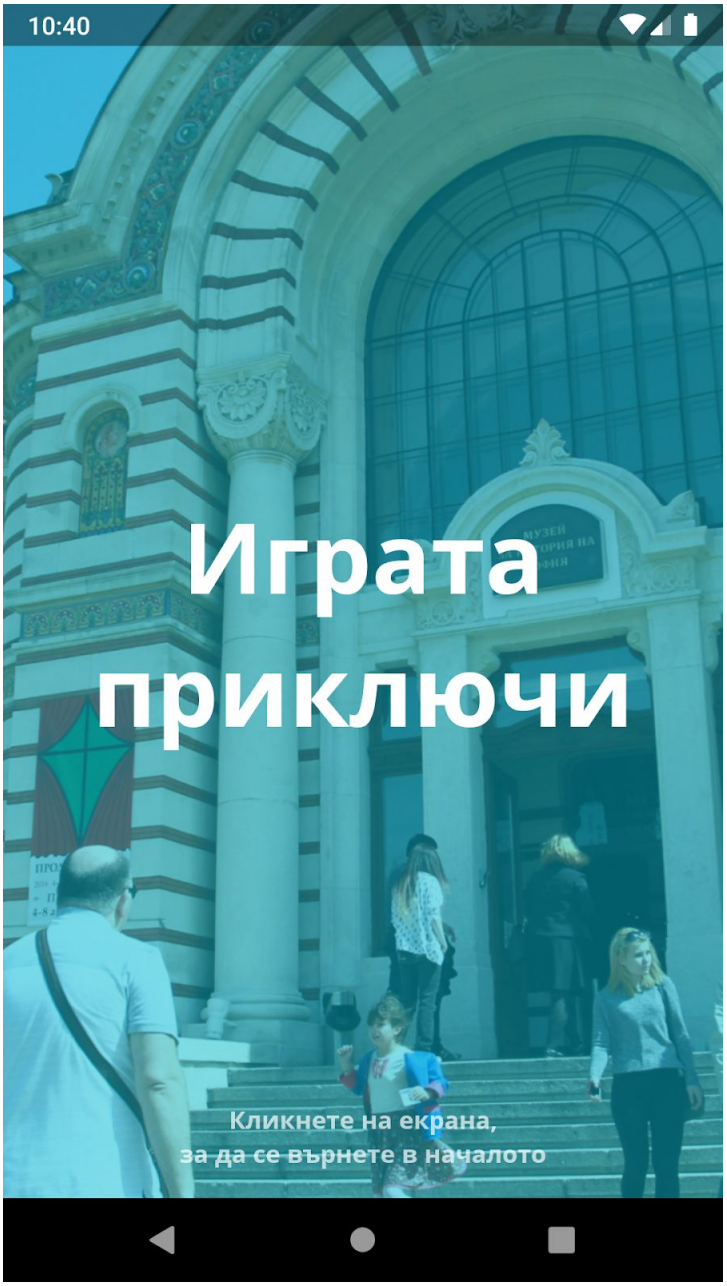


Съдържа:

1. Карта, показваща местоположението на даден музей.
2. Бутон за упътване на играча в играта.
3. Бутон за препращане към Google Maps.
4. При засичане на текущото местоположение, в радиус от 150м. от музея, играта продължава напред.

Екран за край на играта

Този екран обозначава, че играта е приключила.



Съдържа:

1. Надпис, обозначаващ край на играта
2. При натискане където и да е по екрана, потребителят се връща в началото.

Реализация

База данни

Описване на базата

```
@Database(version = DB_VERSION, entities = [Game::class,
LevelProgress::class])
@TypeConverters(
    value = [
        DateTypeConverters::class,
        LevelStageTypeConverters::class,
        StringListTypeConverters::class
    ]
)
abstract class MuseumHuntRoomDB : RoomDatabase() {

    abstract fun gameDao(): GameDao

    abstract fun levelDao(): LevelProgressDao

}
```

Описване на обектите за достъп до базата

```
@Dao
interface GameDao {

    @Query("SELECT * FROM Games")
    fun getAllGames(): Single<List<Game>>

    @Query("SELECT * FROM Games WHERE id = :gameId")
    fun getGameById(gameId: Long): Single<Game?>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertNewGame(game: Game): Single<Long>

}
```

```
@Dao
interface LevelProgressDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertLevelProgress(levelProgress: LevelProgress): Single<Long>

    @Query("UPDATE LevelProgress SET stage = :stage WHERE id = :levelId")
```

```
    fun updateLevelStage(levelId: Long, stage: LevelStage): Completable

    @Update
    fun updateLevelProgress(levelProgress: LevelProgress): Completable

    @Query("SELECT * FROM LevelProgress WHERE id = :levelId")
    fun getLevelProgressById(levelId: Long): Single<LevelProgress>
}
```

Обектите в базата

```
@Entity(
    tableName = TableConstants.GAMES,
    foreignKeys = [
        ForeignKey(
            entity = LevelProgress::class,
            onDelete = ForeignKey.CASCADE,
            onUpdate = ForeignKey.CASCADE,
            parentColumns = ["id"],
            childColumns = ["levelId"]
        )],
    indices = [Index("levelId", unique = true)]
)
data class Game(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val name: String,
    val lastModified: Date,
    val levelId: Long
)
```

```
@Entity(tableName = TableConstants.LEVEL_PROGRESS)
data class LevelProgress(
    @PrimaryKey(autoGenerate = true) val id: Long = 0,
    val number: Int,
    val stage: LevelStage,
    val hintWords: MutableList<String> = mutableListOf()
)

enum class LevelStage {
    INIT, OBSTACLE_PASSED, RIDDLE_PASSED, COMPLETED
}
```

Достъп до базата чрез хранилище (repository)

```
interface GameRepository {
   fun getGames(): Single<List<Game>>

   fun addGame(game: Game): Single<Long>

   fun getGameById(gameId: Long): Single<Game?>

   fun updateLevelProgress(levelProgress: LevelProgress): Completable

   fun setLevelStage(levelId: Long, levelStage: LevelStage): Completable

   fun addLevelProgress(levelProgress: LevelProgress): Single<Long>

   fun getLevelProgressById(levelId: Long): Single<LevelProgress>
}

class GameRepositoryImpl(
   private val gameDao: GameDao,
   private val levelProgressDao: LevelProgressDao
) : GameRepository {

   override fun getGames(): Single<List<Game>> {
       return gameDao.getAllGames()
           .subscribeOn(Schedulers.io())
   }

   override fun addGame(game: Game) =
gameDao.insertNewGame(game).subscribeOn(Schedulers.io())

   override fun getGameById(gameId: Long): Single<Game?> =
gameDao.getGameById(gameId).subscribeOn(Schedulers.io())

   override fun updateLevelProgress(levelProgress: LevelProgress) =

levelProgressDao.updateLevelProgress(levelProgress).subscribeOn(Schedulers.i
o())

   override fun setLevelStage(levelId: Long, levelStage: LevelStage) =
       levelProgressDao.updateLevelStage(levelId,
levelStage).subscribeOn(Schedulers.io())

   override fun addLevelProgress(levelProgress: LevelProgress): Single<Long>
=

levelProgressDao.insertLevelProgress(levelProgress).subscribeOn(Schedulers.i
```

```
o())

   override fun getLevelProgressById(levelId: Long): Single<LevelProgress> =

levelProgressDao.getLevelProgressById(levelId).subscribeOn(Schedulers.io())
}
```

Пример за достъп до данни чрез UseCase:

```
class UpdateLevelStageUseCase @Inject constructor(private val
gameRepository: GameRepository) {

   fun updateLevelStage(levelId: Long, levelStage: LevelStage) =
gameRepository.setLevelStage(levelId, levelStage)

}
```

```
class SetObstaclesPassedUseCase @Inject constructor(
   private val gameRepository: GameRepository,
   private val getLevelProgressDataUseCase: GetLevelProgressDataUseCase,
   private val updateLevelStageUseCase: UpdateLevelStageUseCase
) {

   fun setObstaclesPassedUseCase(levelId: Long, foundWords: List<String>):
Completable =
       updateLevelStageUseCase.updateLevelStage(levelId,
LevelStage.OBSTACLE_PASSED)
           .observeOn(Schedulers.computation())
           .toSingle()
           .flatMap {

getLevelProgressDataUseCase.getLevelProgressDataUseCase(levelId)
           }
           .flatMap {
               it.hintWords.clear()
               it.hintWords.addAll(foundWords)
               gameRepository.updateLevelProgress(it).toSingle()
           }
           .ignoreElement()
}
```

Следене на местоположение:

```
class TrackDestinationReachedUseCase @Inject constructor(private val
locationService: LocationService) {

   private val locationRequest = LocationRequest.create().apply {
       interval = 1000
       fastestInterval = 1000
       priority = LocationRequest.PRIORITY_HIGH_ACCURACY
   }

   @RequiresPermission(anyOf = ["android.permission.ACCESS_COARSE_LOCATION",
"android.permission.ACCESS_FINE_LOCATION"])
   fun startTracking(
       destinationLocation: StageLocation,
       locationUpdateListener: MutableLiveData<Location>
   ): Single<Either<LocationError, LocationData>> {
       return locationService.requestUpdates(locationRequest)
           .observeOn(Schedulers.computation())
           .onUpdate {
               checkDestinationReached(it,
destinationLocation.locationCoordinates)
           }
           .observeOn(AndroidSchedulers.mainThread())
           .onUpdate {
               Timber.d("location updated: ${it.location.latitude} |
${it.location.longitude}")
               locationUpdateListener.postValue(it.location)
           }
           .takeUntil { either ->
               when (either) {
                   is Either.Left<LocationError> -> false
                   is Either.Right<LocationData> -> checkDestinationReached(
                       either.b,
                       destinationLocation.locationCoordinates
                   )
               }
           }
           .onErrorReturn { Either.left(UnknownError(it)) }
           .lastOrError()
           .doOnSubscribe {
               Timber.d("Location updates started")
           }
   }

   private fun checkDestinationReached(
       currentLocation: LocationData,
       locationCoordinates: LocationCoordinates
```

```
    ): Boolean {
        return locationCoordinates.let {
            val results = floatArrayOf(0f)

            Location.distanceBetween(
                currentLocation.location.latitude,
                currentLocation.location.longitude,
                it.latitude,
                it.longitude,
                results
            )

            val distance = results[0]
            Timber.d("Distance to destination: $distance")
            distance <= minDistanceToDestination
        }
    }
}
```

Пример за ViewModel

```
class RiddleViewModel @Inject constructor(
    resourceManager: ResourceManager,
    private val getLevelProgressDataUseCase: GetLevelProgressDataUseCase,
    private val levelDataUseCase: GetLevelDataUseCase,
    private val setRiddleAnsweredUseCase: SetRiddleAnsweredUseCase
) : BaseViewModel(resourceManager) {

    private lateinit var ingameArgs: IngameArgs
    private var levelProgress: LevelProgress? = null

    private val _answersEvent = LiveEvent<List<Answer>>()
    private val _goBackEvent = LiveEvent<Unit>()
    private val _errorSnackBarEvent = LiveEvent<Unit>()
    private val _nextScreenEvent = LiveEvent<IngameArgs>()
    private val _showDialogEvent = LiveEvent<DialogValues>()

    val answersEvent: LiveData<List<Answer>> = _answersEvent
    val goBackEvent: LiveData<Unit> = _goBackEvent
    val errorSnackBarEvent: LiveData<Unit> = _errorSnackBarEvent
    val nextScreenEevent: LiveData<IngameArgs> = _nextScreenEvent
    val showDialogEvent: LiveData<DialogValues> = _showDialogEvent

    @get:Bindable
    var riddle: String? = null
        set(value) {
            field = value
```

```
            notifyPropertyChanged(BR.riddle)
        }

    fun init(ingameArgs: IngameArgs) {
        this.ingameArgs = ingameArgs

getLevelProgressDataUseCase.getLevelProgressDataUseCase(ingameArgs.levelId)
            .observeOn(AndroidSchedulers.mainThread())
            .doOnSuccess {
                levelProgress = it
            }
            .observeOn(Schedulers.computation())
            .flatMap {
                levelDataUseCase.getLevelData(it.number)
            }
            .map { it.stageRiddle }
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(Consumer {
                _answersEvent.postValue(it.answers)
                riddle = it.riddle
            })
            .addTo(container)
    }

    fun onToolbarNavigationClick() {
        _goBackEvent.value = Unit
    }

    fun onHelpClick() {
        _showDialogEvent.value = DialogValues().apply {
            title = getString(R.string.dialog_riddle_hint_title)
            message = levelProgress?.hintWords?.joinToString(", ")
            positiveBtnTxt = getString(R.string.ok)
        }
    }

    fun onAnswerClick(answer: Answer) {
        if (!answer.isCorrect) {
            _errorSnackBarEvent.value = Unit
        } else {
            setRiddleAnsweredUseCase.setRiddleAnswered(ingameArgs.levelId)
                .subscribeOn(AndroidSchedulers.mainThread())
                .subscribe {
                    _nextScreenEvent.postValue(ingameArgs)
                }
                .addTo(container)
        }
    }
```

```
}
```

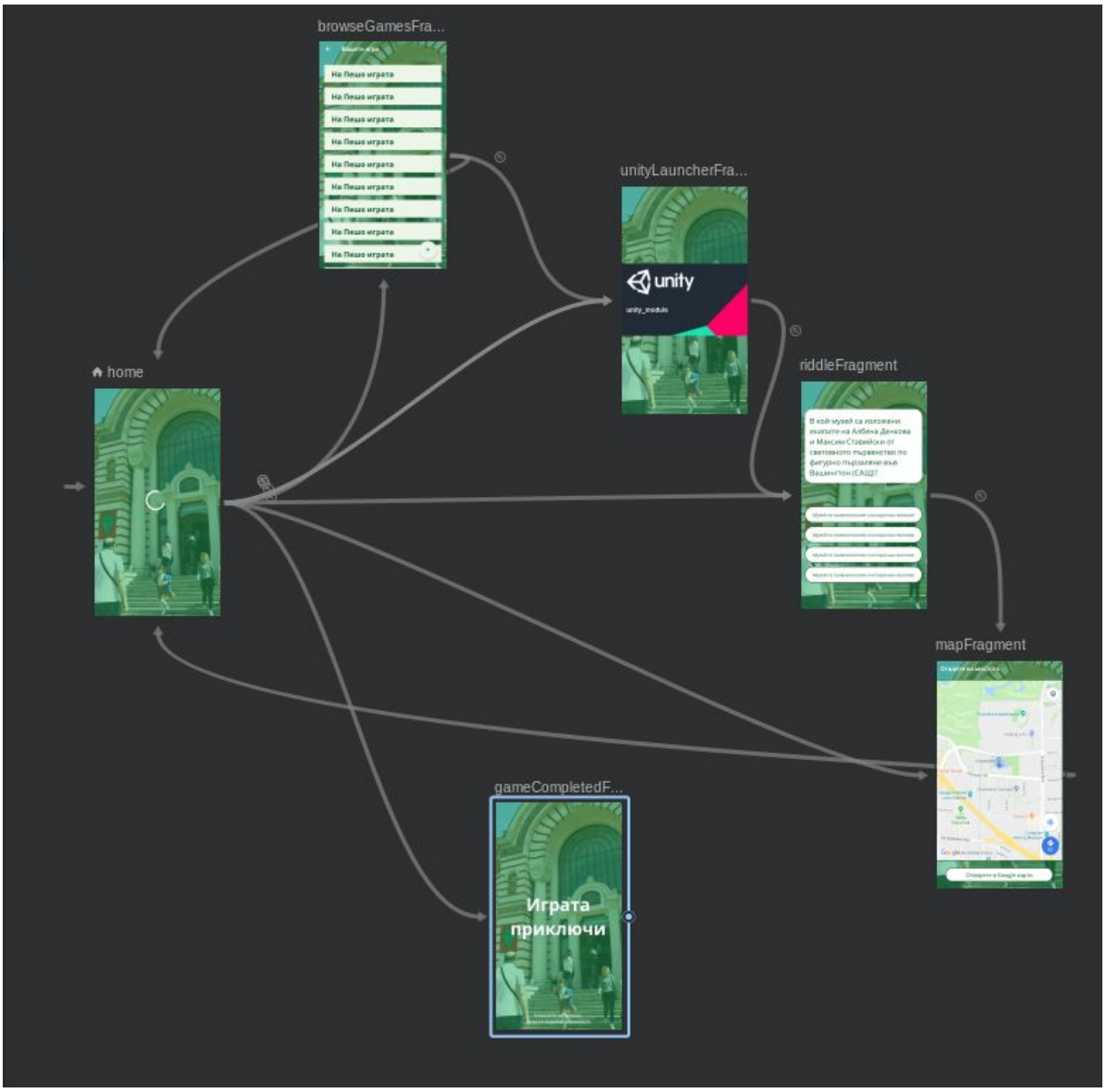
Пример за Fragment:

```
abstract class BaseFragment<BindingT : ViewDataBinding, ViewModelT :
BaseViewModel> : DaggerFragment() {

   @Inject
   lateinit var viewModelFactory: ViewModelProvider.Factory

   protected lateinit var viewModel: ViewModelT

   protected val binding: BindingT? get() = view?.let {
DataBindingUtil.findBinding(it) }

   override fun onCreate(savedInstanceState: Bundle?) {
       performInjection()
       super.onCreate(savedInstanceState)

       viewModel = instantiateViewModel()
   }

   override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
       super.onViewCreated(view, savedInstanceState)
       viewLifecycleOwner.lifecycle.addObserver(viewModel)
   }

   protected open fun performInjection() {
       AndroidSupportInjection.inject(this)
   }

   abstract fun instantiateViewModel(): ViewModelT

   protected fun binding(block: BindingT.(vm: ViewModelT) -> Unit) {
       binding?.block(viewModel)
   }

   protected fun Intent.start() {
       startActivity(this)
   }

   protected fun <T> LiveData<T>.observe(observer: Observer<T>) =
observe(viewLifecycleOwner, observer)
}
```

```
class RiddleFragment : BaseFragment<FragmentRiddleBinding,
RiddleViewModel>() {

   private val input by navArgs<RiddleFragmentArgs>()

   override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?): View? {
       return FragmentRiddleBinding.inflate(inflater, container, false).root
   }

   override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
       super.onViewCreated(view, savedInstanceState)

       finishOnBackPressed()

       val answersAdapter = AnswersAdapter(Observer {
           viewModel.onAnswerClick(it)
       })

       viewModel.answersEvent.observe(viewLifecycleOwner, answersAdapter)

       viewModel.goBackEvent.observe(viewLifecycleOwner, Observer {
           requireActivity().onBackPressedDispatcher.onBackPressed()
       })

       viewModel.errorSnackBarEvent.observe(viewLifecycleOwner, Observer {
           Snackbar.make(view, getString(R.string.riddle_answer_error),
Snackbar.LENGTH_LONG)
               .show()
       })

       viewModel.showDialogEvent.observe(viewLifecycleOwner, Observer {
           AlertDialog.Builder(context, R.style.AppTheme_Dialog_InGame)
               .setTitle(it.title)
               .setMessage(it.message)
               .setPositiveButton(it.positiveBtnTxt) { _, _ ->
it.positiveBtnCallback }
               .show()
       })

       viewModel.nextScreenEevent.observe(viewLifecycleOwner, Observer {
args ->
           args?.let {
findNavController().navigate(RiddleFragmentDirections.actionRiddleFragmentTo
MapFragment(it)) }
       })

       binding { vm ->
```

```
            viewModel = vm

            toolbar.inflateMenu(R.menu.menu_toolbar_riddle_fragment)
            toolbar.setOnMenuItemClickListener {
                if (it.itemId == R.id.menu_item_help) {
                    vm.onHelpClick()
                }

                true
            }

            toolbar.setNavigationIcon(R.drawable.ic_arrow_back)
            toolbar.setNavigationOnClickListener {
                vm.onToolbarNavigationClick()
            }

            rvAnswers.layoutManager = LinearLayoutManager(context)
            rvAnswers.adapter = answersAdapter
        }

        viewModel.init(input.args)
    }

    override fun instantiateViewModel(): RiddleViewModel =
        ViewModelProviders.of(this,
viewModelFactory)[RiddleViewModel::class.java]

}
```

Навигация с Android Navigation Component:



Основно управление в Unity3d модула

```
public class GameController : MonoBehaviour
{
   private static readonly int ID_QUIT_CONFIRMATION_DIALOG = 0;
   private static readonly int ID_HELPDIALOG = 1;

   public GameObject LevelRepositoryObject;
   public GameObject BackButton;
   public GameObject HelpButton;
   public GameObject TextPanel;
   public GameObject Player;

   private TextPanelBinding TextPanelBinding;

   private LevelDataRepository LevelDataRepository;

   //private List<string> LetterPlaceHolders;
   private ObservableCollection<string> SelectedLettersObservable = new
ObservableCollection<string>();
   private Stack<LetterTileBinding> SelectedTilesStack = new
Stack<LetterTileBinding>();

   private ISet<string> WordsSet;
   private int LongestWordLength;
   private ISet<string> FoundWords = new HashSet<string>();

   // Start is called before the first frame update
   void Start()
   {
       var tilesLayout =
GameObject.Find("TilesLayout").GetComponent<TilesLayout.TilesLayout>();

       SelectedLettersObservable.CollectionChanged += (sender, args) =>
       {
           TextPanelBinding.Text = string.Join("", SelectedLettersObservable);
       };

       tilesLayout.OnGameObjectClick += (gameObject) =>
       {
           var tileBinding = gameObject.GetComponent<LetterTileBinding>();
           if (tileBinding != null)
           {
               OnTileClick(tileBinding);
           }
       };

       LevelDataRepository =
LevelRepositoryObject.GetComponent<LevelDataRepository>();
       TextPanelBinding = TextPanel.GetComponent<TextPanelBinding>();
```

```
        var hintWords = LevelDataRepository.GetLevelData().HintWords;
        WordsSet = hintWords.Select(word => word.ToUpper()).ToSet();
        LongestWordLength = (from word in hintWords
                             orderby word.Length
                             select word.Length).Last();

        GameObject.FindGameObjectWithTag("Finish")
            .GetComponent<Level.FinishLineWatcher>()
            .OnPass.AddListener(OnGameFinish);

        ShowHelpDialog();
    }

    void Update()
    {
        if (Input.GetKeyUp(KeyCode.Escape))
        {
            OnBackPressed();
        }
    }

    protected void OnBackPressed()
    {
        if (MsgBox.isOpen)
        {
            MsgBox.Close();
        } else
        {
            OnBackClick();
        }
    }

    private IEnumerable<string> CreateEmpty(int size)
    {
        for (int i = 0; i < size; i++)
        {
            yield return null;
        }
    }

    public void OnBackClick()
    {
        new MsgBox.Builder
        {
            id = ID_QUIT_CONFIRMATION_DIALOG,
            message = "Нивото няма да бъде
запазено.\nСигурен/а ли сте, че искате да
излезете.",
            negativeCallback = id =>
            {
                MsgBox.Close();
```

```
            },
            negativeButtonText = "Не",
            positiveCallback = id =>
            {
                MobileApplication.Quit();
            },
            positiveButtonText = "Да",
            style = MsgBoxStyle.Information,
            buttons = MsgBoxButtons.YES_NO,
            modal = true
        }
       .Show();
    }

    public void OnHelpClick()
    {
        ShowHelpDialog();
    }

    private void ShowHelpDialog()
    {
        new MsgBox.Builder
        {
            id = ID_HELPDIALOG,
            message = "Минете нивото, за да отключите
загадка.",
            positiveCallback = id =>
            {
                MsgBox.Close();
            },
            style = MsgBoxStyle.Information,
            buttons = MsgBoxButtons.OK,
            modal = true,
            positiveButtonText = "Добре"
        }
        .Show();
    }

    public void OnClearClick()
    {
        ClearSelection();
    }

    public void OnGameFinish()
    {
        Destroy(Player);
        MobileApplication.Quit(new ResultData(FoundWords.ToArray()));
    }

    private void OnTileClick(LetterTileBinding binding)
    {
```

```
        if ((binding.TileState == LetterTileBinding.State.SELECTED ||
binding.TileState == LetterTileBinding.State.ERROR)
            && SelectedTilesStack.Peek() != binding) return;

        var currentWord = string.Join("", SelectedLettersObservable);
        string newWord;

        if (binding.TileState == LetterTileBinding.State.IDLE)
        {
            SelectedTilesStack.Push(binding);
            SelectedLettersObservable.Add(binding.Letter);
            newWord = currentWord + binding.Letter;
        }
        else
        {
            var popped = SelectedTilesStack.Pop();
            popped.TileState = LetterTileBinding.State.IDLE;
            SelectedLettersObservable.RemoveAt(SelectedLettersObservable.Count -
1);
            newWord = currentWord.Substring(0, currentWord.Length - 1);
        }

        switch (CheckWord(newWord))
        {
            case WordCheckResult.CONTAINS:
                SelectedTilesStack.Apply(tile => tile.TileState =
LetterTileBinding.State.SELECTED);
                break;
            case WordCheckResult.MATCH:
                SelectedTilesStack.Apply(tile => tile.TileState =
LetterTileBinding.State.INACTIVE);
                WordsSet.Remove(newWord);
                ClearSelection();
                FoundWords.Add(newWord);
                break;
            case WordCheckResult.ERROR:
                SelectedTilesStack.Apply(tile => tile.TileState =
LetterTileBinding.State.ERROR);
                break;
        }
    }

    private void ClearSelection()
    {
        SelectedLettersObservable.Clear();
        SelectedTilesStack.Apply(binding =>
        {
            if (binding.TileState != LetterTileBinding.State.INACTIVE)
            {
                binding.TileState = LetterTileBinding.State.IDLE;
            }
```

```
        });
        SelectedTilesStack.Clear();
    }

    private LetterTileBinding.State GetStateForResult(WordCheckResult result)
    {
        LetterTileBinding.State state;
        switch (result)
        {
            case WordCheckResult.CONTAINS:
            case WordCheckResult.MATCH:
                state = LetterTileBinding.State.SELECTED;
                break;
            case WordCheckResult.ERROR:
                state = LetterTileBinding.State.ERROR;
                break;
            default:
                throw new InvalidOperationException($"Could not handle unknown
result: {result}");
        }

        return state;
    }

    private WordCheckResult CheckWord(string queryWord)
    {
        if (WordsSet.Contains(queryWord))
        {
            return WordCheckResult.MATCH;
        }

        if (queryWord.Length > LongestWordLength)
        {
            return WordCheckResult.ERROR;
        }

        if (WordsSet.Any(word => word.StartsWith(queryWord)))
        {
            return WordCheckResult.CONTAINS;
        }

        return WordCheckResult.ERROR;
    }

    private enum WordCheckResult
    {
        CONTAINS, MATCH, ERROR
    }
}
```

Заключение

“Лов на музеи” е работещ продукт, който ангажира вниманието на любознателните. Простият интерфейс позволява удобство при игра.

Приложението е изпробвано на различни резолюции и успешно скалира интерфейса по тях.

Заради природата на Unity3d, размерът на приложението е по-голям от необходимото, затова има място за подобрение от този аспект.

Архитектурата на проекта позволява лесно модифициране и надграждане.

Код на проекта

Проектът може да бъде клониран от едно от двете публични хранилища:

- GitHub - https://github.com/bkolarov/museum_hunt.git
- BitBucket - https://bogikolarov@bitbucket.org/bogikolarov/bg_quest.git

Литература

- Android Room - https://developer.android.com/jetpack/androidx/releases/room
- Navigation Component - https://developer.android.com/guide/navigation
- Dagger 2 - https://google.github.io/dagger/
- Google Maps - https://developers.google.com/maps/documentation/android-sdk/intro
- Google APIs - https://developers.google.com/android/
- RxJava 2 - http://reactivex.io/
- Unity3d - https://docs.unity3d.com/Manual/index.html
- Android SDK - https://developer.android.com/docs