



دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

پایان‌نامه‌ی کارشناسی
گرایش مهندسی نرم‌افزار

عنوان:

طراحی میان‌افزار برای اتصال حسگرهای IoT در بازی‌های
واقعیت فرآگیر

نگارش:

شمیم کریمی

استاد راهنما:

دکتر علی‌اصغر نظری شیره‌جینی

۱۳۹۶ تیر

مَلَكُ الْأَفْلَكَ

به نام خدا
دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

پایان‌نامه‌ی کارشناسی

عنوان: طراحی میان‌افزار برای اتصال حسگرهای IoT در بازی‌های واقعیت فراگیر
نگارش: شمیم کریمی

کمیته‌ی ممتحنین

استاد راهنما: دکتر علی‌اصغر نظری امضاء:
شیره‌جینی

استاد مشاور: استاد مشاور امضاء:

استاد مدعو: استاد ممتحن امضاء:

تاریخ:

فهرست مطالب

| | | |
|----|-------|------------------------------|
| ۶ | ۱ | مقدمه |
| ۷ | ۱-۱ | تعریف مسئله |
| ۹ | ۱-۱-۱ | سناریوی نمونه |
| ۱۴ | ۲-۱ | اهمیت موضوع |
| ۱۶ | ۳-۱ | اهداف تحقیق |
| ۱۷ | ۲ | کارهای پیشین |
| ۲۰ | ۳ | معماری سامانه |
| ۲۳ | ۴ | بررسی تکنولوژی‌های موجود |
| ۲۷ | ۵ | ماژول‌ها |
| ۲۷ | ۱-۵ | ماژول تشخیص فعالیت‌های کاربر |
| ۲۷ | ۱-۱-۵ | معرفی ماژول |
| ۲۸ | ۲-۱-۵ | نمونه‌ای از کارکرد برنامه |
| ۳۰ | ۳-۱-۵ | نمودارهای تحلیل و طراحی |
| ۳۵ | ۴-۱-۵ | پیاده‌سازی |

| | | |
|----|-------|---------------------------------|
| ۴۳ | | ۲-۵ مژول خواندن تگ NFC |
| ۴۳ | | ۱-۲-۵ معرفی مژول |
| ۴۳ | | ۲-۲-۵ نمونه‌ای از کارکرد برنامه |
| ۴۷ | | ۳-۲-۵ نمودارهای تحلیل و طراحی |
| ۵۰ | | ۴-۲-۵ پیاده‌سازی |
| ۵۴ | | ۳-۵ مژول خواندن تگ QRCode |
| ۵۴ | | ۱-۳-۵ معرفی مژول |
| ۵۴ | | ۲-۳-۵ نمونه‌ای از کارکرد برنامه |
| ۵۶ | | ۳-۳-۵ نمودارهای تحلیل و طراحی |
| ۵۸ | | ۴-۳-۵ پیاده‌سازی |
| ۶۱ | | ۴-۵ مژول گام‌شمار |
| ۶۱ | | ۱-۴-۵ معرفی مژول |
| ۶۱ | | ۲-۴-۵ نمونه‌ای از کارکرد برنامه |
| ۶۲ | | ۳-۴-۵ نمودارهای تحلیل و طراحی |
| ۶۴ | | ۴-۴-۵ پیاده‌سازی |
| ۶۶ | | ۵-۵ مژول شمارندهی حرکت پروانه |
| ۶۶ | | ۱-۵-۵ معرفی مژول |
| ۶۶ | | ۲-۵-۵ نمونه‌ای از کارکرد برنامه |
| ۶۷ | | ۳-۵-۵ نمودارهای تحلیل و طراحی |
| ۶۹ | | ۴-۵-۵ پیاده‌سازی |

فصل ۱

مقدمه

با پیشرفت تکنولوژی در زمینه‌ی هوشمندی محیط، باب جدیدی در برخی مفاهیم گذشته باز شده است. یکی از این زمینه‌ها، طراحی و توسعه‌ی بازی‌های رایانه‌ای است. با استفاده از تکنولوژی هوشمندی محیط، امکان طراحی و تولید بازی‌هایی به وجود آمده است که به دنیای واقعی و یا دنیای رایانه محدود نمی‌شوند؛ بلکه می‌توان بازی را به گونه‌ای تعریف کرد که تعامل در هر دو دنیای واقعی و مجازی وجود داشته باشد. این نوع از بازی‌ها که به نام بازی‌های واقعیت فراگیر و یا واقعیت آغشته شناخته می‌شوند، گونه‌ای نوین از بازی‌ها هستند که از مرز بین دنیای واقعی و مجازی عبور می‌کنند. به این معنی که تاثیرات تعامل در یک دنیا، در دنیای دیگر ادامه می‌یابد و رد پای فعالیت‌های انجام‌شده در یک دنیا، در هر دو دنیا مشهود است. اهمیت این نوع بازی از این جهت است که اثرگذاری بر کاربر را به بیشترین حد خود می‌رساند. چرا که فعالیت‌هایی که تمام حواس را درگیر کنند، به واقعیت نزدیک باشند و نیاز به تعامل بیشتر داشته باشند، باعث افزایش تاثیرگذاری بر کاربر می‌شوند. به همین دلیل انتقال مفاهیم از طریق این گونه بازی‌ها می‌تواند روشی نوین در آموزش و پرورش هرچه بهتر نسل‌های آینده باشد.

۱-۱ تعریف مسئله

بازی‌های واقعیت فراگیر گونه‌ای از بازی‌ها هستند که بین دنیای واقعی و دنیای مجازی پل ارتباطی برقرار می‌کنند. در این بازی‌ها کاربر می‌تواند هم از طریق انواع رایانه در دنیای مجازی به بازی بپردازد، و هم می‌تواند در یک محیط واقعی و به دور از صفحه‌ی رایانه بازی را دنبال کند. بازی باید به گونه‌ای طراحی شود که فعالیت‌های کاربر بر شرایط هر دو دنیا تاثیر بگذارد، و در هر لحظه شرایط هر دو دنیا با هم هماهنگ باشد. به طوری که کاربر بتواند بازی در یک محیط را رها کرده و در دنیای دیگر به ادامه‌ی آن بازی بپردازد. به این ترتیب لازم است که طراح بازی معماری سامانه را با توجه به ویژگی‌های زیر طراحی کند:

- یک نکته‌ی اساسی در طراحی بازی‌های واقعیت فراگیر، طراحی نحوه‌ی تعامل کاربر با بازی است. اگر بازی را به دو بخش کلی دنیای واقعی و دنیای مجازی تفکیک کنیم، می‌توان تعامل در هر بخش را با استفاده از تکنولوژی‌های زیر ایجاد کرد:

- دنیای واقعی: استفاده از حسگرهای مختلف، تشخیص موقعیت‌ها و فعالیت‌های کاربر در محیط واقعی را ممکن می‌سازد. همچنین استفاده از عملگر یا Actuator می‌تواند تغییرات مورد نیاز را در شرایط محیط ایجاد کند. در ادامه‌ی تعریف مسئله مثال‌هایی از این حسگرهای عملگرها آورده می‌شود.

- دنیای مجازی: تعامل در دنیای مجازی به دستگاه مورد استفاده بستگی دارد و به صورت‌های مختلفی امکان‌پذیر است. انواع دستگاه‌هایی که برای بازی‌های رایانه‌ای استفاده می‌شوند می‌توانند در این نوع بازی‌ها نیز مورد استفاده قرار بگیرند. رایانه‌ی شخصی، تلفن همراه هوشمند، کنسول‌های بازی و ... از انواع این دستگاه‌ها به شمار می‌روند.

- این بازی می‌تواند به صورت گروهی یا فردی انجام شود و این موضوع به سناریوی پیاده‌سازی شده بستگی دارد. در صورتی که بازی به صورت گروهی باشد، باید قابلیت بازی همزمان توسط چند نفر در هر دو دنیای واقعی و مجازی وجود داشته باشد. برای مثال یک نفر در محیط واقعی به بازی بپردازد، و هم‌گروهی آن شخص از خانه و با استفاده از رایانه او را در بازی همراهی کند. به این ترتیب مسئله‌ی هماهنگ بودن دو دنیا اهمیت بیشتری پیدا می‌کند.

- در این بازی، هر شیء یا مفهوم در دنیای واقعی، یک همتا در دنیای مجازی دارد. البته با توجه به

تفاوت ماهیت دو دنیا، منطقی است اگر نحوه تعامل با این شیء یا مفهوم در دو دنیا متفاوت باشد.

• طراحی سناریو نقش مهمی در این نوع بازی ایفا می‌کند. روش‌های گوناگونی برای ایجاد هیجان و انگیزه در کاربر با استفاده از امکانات بازی‌های فراگیر وجود دارد. همچنین تاثیرگذاری این تجربه به علت فراگیر بودن تعامل و درگیری تمام حواس کاربر، می‌تواند بیشتر از انواع دیگر بازی‌ها باشد. به همین دلیل درگیر کردن کاربران در سناریوهایی با اهداف آموزشی و تربیتی، یکی از مهم‌ترین نتایج توسعه‌ی این نوع بازی است. علاوه بر اهداف اصلی تعریف شده برای سناریوها، بازی باید جذابیت لازم برای علاقه‌مند کاربر را داشته باشد. به همین دلیل می‌توان طراحی سناریو را یکی از مهم‌ترین قدم‌ها در ساخت این بازی دانست.

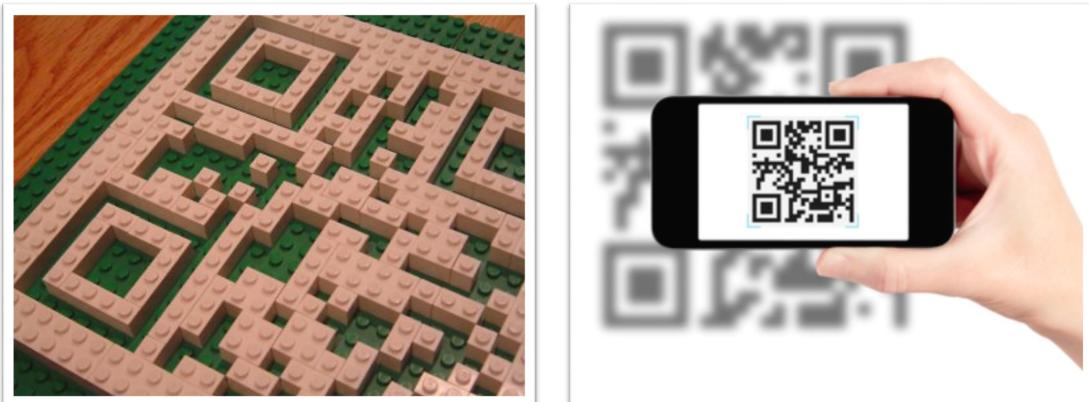
۱-۱-۱ سناریوی نمونه

در این بخش به شرح یک سناریوی نمونه در بازی واقعیت فرآگیر می‌پردازیم. گرچه این سناریوی نمونه تنها بخش کوچکی از کاربردها و قابلیت‌های این نوع بازی را نشان می‌دهد، اما به شفافیت بیشتر مفهوم بازی‌های واقعیت فرآگیر کمک می‌کند.

این سناریو شامل ۴ مرحله یا Level از بازی است. هر یک از این مراحل ارتباط بازی با مأذول‌های طراحی شده در این پروژه را نشان می‌دهد. این مراحل باید به ترتیب توسط کاربر گذرانده شود. با گذراندن هر مرحله، کاربر اطلاعات مورد نیاز برای انجام مرحله‌ی بعدی را کسب می‌کند. مراحل به گونه‌ای طراحی شده‌اند که ترکیبی از بازی‌های فکری و فعالیت‌های بدنی را شامل شوند.

- مرحله‌ی اول: در این مرحله کاربر باید طبق نقشه‌ای که به او داده می‌شود، قطعه‌های کوچک لگو (Lego) را بچیند. چینش این قطعات در نهایت یک تصویر QR Code تشکیل می‌دهد. کاربر باید با استفاده از گوشی همراهی که در اختیار او قرار داده شده، اطلاعات پنهان شده در این QR Code را بخواند.

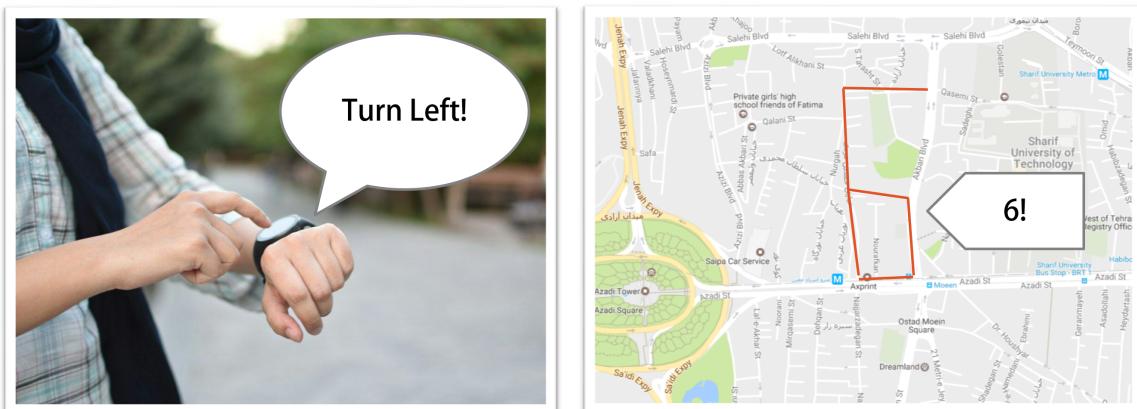
برای پیاده‌سازی این مرحله از لحاظ فنی، کافی است که امکان خواندن QR Code را برای کاربر فراهم آوریم.



شکل ۱-۱: تصویر مرحله‌ی اول بازی: ساخت QR Code و خواندن اطلاعات آن.

• مرحله‌ی دوم: در این مرحله کاربر باید با توجه به دستورالعمل‌هایی که از ساعت هوشمند دریافت می‌کند، مسیر خاصی را پیماید. (اگر محدودیت زمانی نیز برای طی مسیر وجود داشته باشد، کاربر باید این مسیر را بذود.) دستورالعمل‌ها به گونه‌ای به کاربر داده می‌شود که مسیرها شکل خاصی را در نقشه به وجود بیاورند. این شکل به تدریج و با طی مسیرها توسط کاربر، کامل می‌شود. این مرحله زمانی به پایان می‌رسد که کاربر پیغام پنهان‌شده در نقشه را دریابد.

برنامه‌ای که برای این مرحله نیاز داریم، یک برنامه‌ی ساعت هوشمند است که بتواند با استفاده از GPS مکان کاربر روی نقشه را تشخیص دهد. با تشخیص مکان کاربر می‌توان دستورالعمل‌های مربوط به آن مکان را به او اعلام کرد. برای مثال هنگام رسیدن به یک چهارراه، باید به کاربر اعلام شود که برای ادامه‌ی مسیر بازی به سمت چپ بپیچد. قابلیت دیگر این برنامه باید نشان دادن مسیرهای طی شده به کاربر باشد. کاربر با مشاهده‌ی مسیرها روی نقشه می‌تواند پیغام این مرحله از بازی را کشف کند.



شکل ۱-۲: تصویر مرحله‌ی دوم بازی: یافتن پیغام پنهان‌شده در نقشه.

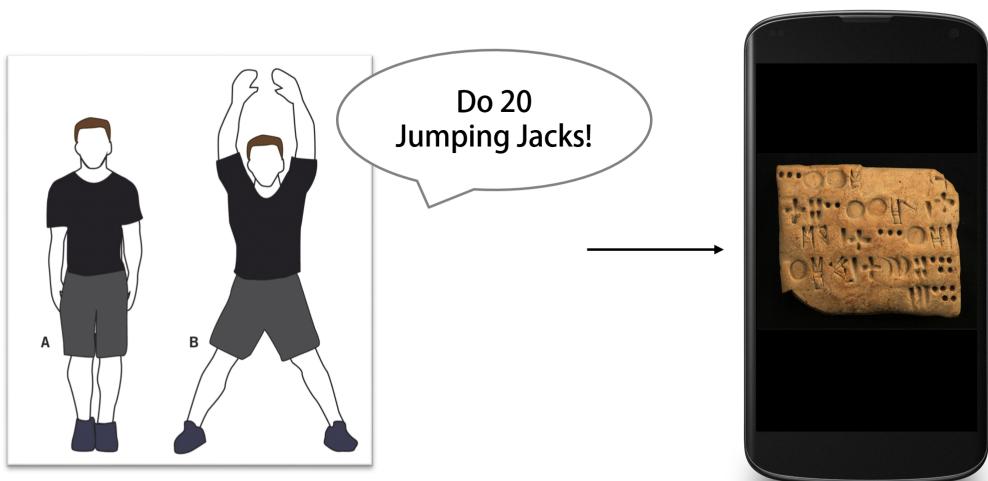
- مرحله‌ی سوم: در مرحله‌ی سوم کاربر باید سه فعالیت بدنی را به مدت مشخص انجام دهد تا یک تصویر به طور کامل به او نشان داده شود. ده دقیقه دوچرخه سواری، پانزده دقیقه دویدن و انجام ۲۰ حرکت پروانه در کمتر از ۳ دقیقه به کاربر کمک می‌کند که مرحله‌ی سوم را پشت سر بگذارد. برای پیاده‌سازی این مرحله از بازی، باید اپلیکیشنی طراحی شود که فعالیت‌های بدنی کاربر را تشخیص دهد. این اپلیکیشن می‌تواند مبتنی بر گوشی هوشمند و یا ساعت هوشمند باشد.



شکل ۱-۳: تصویر مرحله‌ی سوم بازی: ده دقیقه دوچرخه‌سواری.

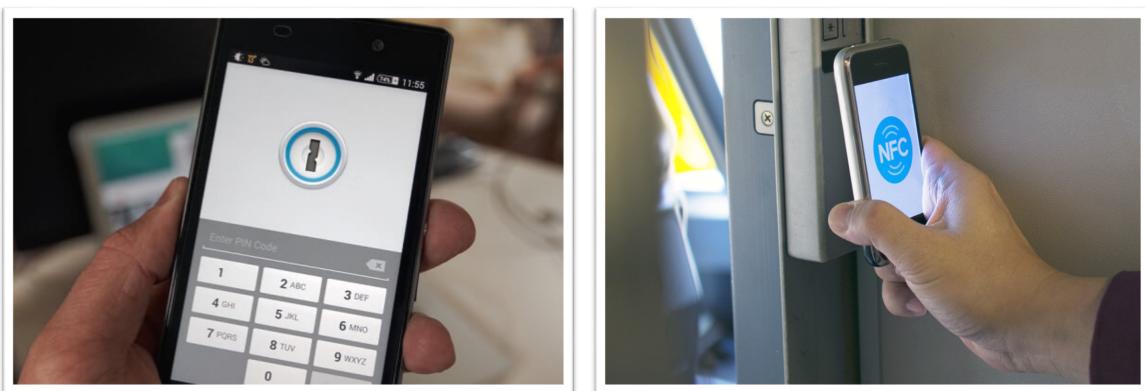


شکل ۱-۴: تصویر مرحله‌ی سوم بازی: پانزده دقیقه دویدن.



شکل ۱-۵: تصویر مرحله‌ی سوم بازی: انجام ۲۰ حرکت پروانه.

- مرحله‌ی چهارم: در این مرحله کاربر باید رمز پیدا شده از مرحله‌ی قبل را برای باز کردن یک در به کار بگیرد. باز کردن در با استفاده از تکنولوژی NFC انجام می‌شود. یک گوشی هوشمند مجهز به NFC می‌تواند هم به عنوان تگ و هم به عنوان خواننده‌ی تگ استفاده شود. در این کاربرد خاص، گوشی هوشمند باید به عنوان تگ عمل کند و قفل در به عنوان خواننده‌ی تگ. به این ترتیب اگر رمز موجود در گوشی صحیح بود، قفل در برای کاربر باز می‌شود.



شکل ۱-۶: تصویر مرحله‌ی چهارم بازی: باز کردن قفل در با استفاده از تکنولوژی NFC.

۲-۱ اهمیت موضوع

بازی، یکی از راههای انتقال مفاهیم به صورت غیرمستقیم به افراد در سنین مختلف است. درگیری کاربران در جریان یک بازی می‌تواند اهداف بلند مدت آموزشی و تربیتی را محقق سازد. بدیهی است که هرچه در طراحی بازی موفق‌تر عمل کنیم، این تاثیرگذاری بیشتر و گستردگر خواهد بود. موفقیت یک بازی و محبوبیت آن در میان کاربران به عوامل مختلفی بستگی دارد و دسته‌بندی‌های گوناگونی برای این عوامل وجود دارد. در اینجا به یک دسته‌بندی که در [۱] آمده است اشاره می‌کنیم. در این دسته‌بندی به چهار جنبه‌ی یک بازی که بر تجربه‌ی کاربری تاثیر می‌گذارد اشاره شده است.

- چالش‌های فیزیکی: احساس کاربر در برخورد با اشیای فیزیکی یا انسان‌ها.
- چالش‌های ذهنی: درگیری ذهنی کاربر و استفاده از هوش و استدلال در بازی مانند حل معماها.
- تجربه‌ی جمعی: تعامل و ارتباط با دیگر افرادی که در بازی حضور دارند. اخیراً توجه ویژه‌ای به این مورد صورت گرفته است چرا که بازی‌های کامپیوترا کاربران را در معرض دچار شدن به مشکلات اجتماعی در برقراری ارتباط با دیگران قرار می‌دهند.
- غوطه‌ور شدن در بازی یا درگیری همه‌جانبه در بازی: ارزیابی این شاخص پیچیده‌تر از شاخص‌های قبلی است، اما در سرگرم‌کننده بودن بازی نقش بسیار مهمی ایفا می‌کند.

حال برای آن که بازی قابلیت سرگرم کردن کاربر و تاثیرگذاری روی او را داشته باشد، باید تمامی موارد ذکر شده را در بر بگیرد. با توجه به تعریفی که از بازی‌های واقعیت فراگیر ارائه دادیم مشخص است که امکان ایجاد تمام چالش‌های گفته شده در این نوع بازی وجود دارد. برای روشن‌تر شدن این موضوع، این شاخص‌ها را در انواع بازی در جدول ۱-۷ مقایسه می‌کنیم.

در این جدول بیشتر بودن تعداد ستاره‌ها به این معنی است که آن چالش نمود بیشتری در آن نوع بازی دارد. طبیعی است که در یک بازی تماماً فیزیکی، چالش‌های فیزیکی بیشتر باشند. و همینطور عکس این قضیه برقرار است؛ در یک بازی تماماً مجازی نمی‌توان چالش‌های فیزیکی ایجاد کرد. اما بازی‌های واقعیت فراگیر بین این دو حالت قرار می‌گیرند و می‌توان حسی از هر دو دنیا و یا ترکیب آن‌ها ایجاد کرد.

در مورد چالش‌های ذهنی، محدودیتی مربوط به نوع دنیا وجود ندارد. گرچه می‌توان گفت ظرفیت

| دنیای مجازی | ترکیب دنیای واقعی و مجازی | دنیای فیزیکی | |
|-------------|---------------------------|--------------|---------------------|
| * | ** | *** | چالش‌های فیزیکی |
| *** | ** | ** | چالش‌های ذهنی |
| * | ** | *** | تجربه‌ی جمعی |
| ** | *** | * | غوطه‌ور شدن در بازی |

شکل ۱-۷: مقایسه‌ی انواع بازی از لحاظ عوامل موثر بر سرگرم‌کنندگی.

بازی‌های مجازی به خاطر وجود عنصر تخیل در این مورد بیشتر است.

اما آخرین شاخص به میزان درگیری حواس کاربر در بازی اشاره دارد. بازی‌های مجازی عموماً نیاز به تمرکز بیشتری نسبت به بازی‌های فیزیکی دارند بنابراین می‌توان گفت که درگیری کاربر در بازی‌های مجازی نسبتاً بیشتر است. از طرف دیگر، مزیت بازی‌های فیزیکی نسبت به بازی‌های مجازی این است که کاربر را محدود به یک مانیتور نمی‌کند و حواس بیشتری را به کار می‌گیرد. اما بازی‌های واقعیت فرآگیر مزایای هر دو دنیا را در خود گنجانده است. بنابراین در این شاخص نسبت به دو نوع دیگر موفق‌تر عمل کرده است.

نتایج این جدول نشان می‌دهد که اگر بخواهیم شاخص‌های "سرگرمی" و "تأثیرگذاری" را بیشینه کنیم، بازی‌های واقعیت فرآگیر انتخاب مناسبی به نظر می‌رسند. بدین ترتیب اهمیت پرداختن به نحوه‌ی طراحی و پیاده‌سازی این نوع بازی مشخص می‌شود.

۳-۱ اهداف تحقیق

هدف از انجام این تحقیق، طراحی و پیاده‌سازی ماژول‌هایی است که می‌توانند در بازی‌های واقعیت فرآگیر به کار گرفته شوند. این ماژول‌ها بیشتر در حوزه‌ی تعامل کاربر با بازی کاربرد دارند و مبتنی بر دستگاه‌های هوشمندی هستند که کاربر در بازی به همراه دارد. طراحی و پیاده‌سازی این امکانات به صورت ماژول به ما کمک می‌کند که سرعت تعریف یک بازی با سناریوی جدید را بالا ببریم. مثال‌هایی از کاربرد این ماژول‌ها در قسمت سناریوی نمونه آورده شده است. فهرست ماژول‌های پیاده‌سازی شده در این تحقیق عبارتند از:

- ماژول تشخیص فعالیت‌های کاربر
- ماژول خواندن تگ NFC
- ماژول خواندن تگ QRCode
- ماژول گام‌شمار
- ماژول شمارنده‌ی حرکت پروانه

۲ فصل

کارهای پیشین

در این فصل به مرور و بررسی فعالیت‌هایی که تا کنون در حوزه‌ی بازی‌های واقعیت فراگیر انجام شده می‌پردازیم.

اصطلاح Pervasive Computing برای نخستین بار در سال ۱۹۹۸ توسط IBM مطرح شد و به معنی یکپارچگی کامپیوترها در محیط اطراف است. اصطلاح مرتبط دیگر که بسیار شنیده می‌شود، Ubiquitous Computing است. این اصطلاح نیز کاربرد گسترده‌ای دارد اما معمولاً برای کاربرد محاسبه‌ی فراگیر در بازی‌ها به کار گرفته نمی‌شود. در واقع Ubiquitous Computing از جنبه‌ی آکادمیک به حضور تکنولوژی در زندگی مردم می‌پردازد، در حالی که Pervasive Computing از جنبه‌ی صنعتی و در ارتباط با اهداف کوتاه مدت در تجارت الکترونیک و فعالیت‌های تجاری مبتنی بر وب مطرح می‌شود. به همین دلیل در ارتباط با بازی‌های واقعیت فراگیر، بیشتر از اصطلاح Pervasive Computing استفاده می‌شود.

همچنین در این جا لازم است به تفاوت دو مفهوم Mixed Reality Games و Pervasive Games اشاره کنیم. هر بازی‌ای که دنیای حقیقی و مجازی را به هم مرتبط می‌سازد را نمی‌توان یک نمونه از Pervasive Game دانست. بلکه وجود جنبه‌ی Pervasive Computing در این نوع بازی حائز اهمیت است. بنابراین می‌توان Pervasive Games را زیرمجموعه‌ای از Mixed Reality Games دانست که علاوه بر ترکیب واقعیت و مجاز، از تکنولوژی‌های Mobile استفاده می‌کنند. [۱]

پس از بررسی تعاریف و اصطلاحات، مطلوب است تقسیم‌بندی مناسبی از حوزه‌های مرتبط با بازی‌های واقعیت فراگیر داشته باشیم و به تحلیل کارهای پیشین در هریک بپردازیم.

نخستین نکته‌ای که در طراحی و ساخت بازی اهمیت می‌یابد، هدف از ساخت آن بازی است. هرچند

عنصر سرگرمی از بازی جدایی‌ناپذیر است، اما ممکن است هدف اصلی ساخت بازی نباشد. یکی از جنبه‌های بازی‌های واقعیت فرآگیر که بسیار مورد توجه قرار گرفته است، کاربرد آن در حوزه‌ی آموزش است. مقاله‌ی [۲] با استفاده از متداول‌وزی Case Study به معرفی یک چارچوب راهنمایی برای طراحی بازی‌های واقعیت فرآگیر آموزشی پرداخته است. در این تحقیق، سه گروه با پیش‌زمینه‌های کاری متفاوت، یک بازی طراحی شده را ارزیابی کرده و مشکلات آن را گزارش کردند. نتیجه‌ی این ارزیابی در قالب یک چارچوب (Guidline) آمده است. نکته‌ی حائز اهمیت وجود مرحله‌ای برای دخیل کردن جنبه‌های آموزشی در بازی است. پیشنهادات مطرح شده در این مرحله شامل این موارد است: انجام یک فعالیت مرتبط با مباحث آموزشی پیش از شروع بازی، برقراری ارتباط بین اجزای مختلف بازی مانند مکان‌ها و اشیاء توسط کاربر برای درک مفاهیم آموزشی، و برقراری تعادل میان میزان محتوای آموزشی و محتوای سرگرم‌کننده.

همچنین در مواردی به صورت متمرکز به طراحی بازی در آموزش حوزه‌های مشخص پرداخته شده است. در مقاله‌ی [۳] یک بازی طراحی و معرفی شده است که برای آشنایی دانشجویان رشته‌ی فناوری رسانه با مفاهیم پایه‌ی این حوزه مفید است. به طور مشخص معرفی تکنولوژی‌های وب و تیم‌سازی دو هدف اصلی این بازی هستند. مقاله‌ی [۴] از طراحی یک بازی مبتنی بر گوشی همراه هوشمند که در مراکز شهرها قابل استفاده است سخن گفته است. اهداف آموزشی این بازی به شرح زیر عنوان شده است: مهارت‌های مورد نیاز جامعه در قرن بیست و یکم، مانند تفکر انتقادی، خلاقیت، همکاری، در نظر گرفتن جنبه‌های مختلف مرتبط به یک موضوع، بیداری اجتماعی، مسئولیت‌های اجتماعی و تاثیرات رسانه. این موارد نشان می‌دهد که محتوای آموزشی به آموزش‌های آکادمیک محدود نمی‌شود و می‌تواند به مفاهیم اجتماعی و فرهنگی نیز گسترش پیدا کند. در نمونه‌ی مشابه دیگر، مقاله‌ی [۵] یک بازی برای اصلاح الگوی مصرف انرژی در خانه طراحی کرده است.

نکته‌ی دیگر، چگونگی طراحی معماری بازی است. در واقع تعریف مولفه‌ها و چگونگی ارتباط آن‌ها با یکدیگر به نوع بازی و همچنین تکنولوژی‌های انتخاب‌شده بستگی دارد. مقاله‌ی [۶] این نوع بازی را نوعی از سامانه‌های Context Aware در نظر گرفته است. در این معماری پیشنهادی، ارتباط بین مولفه‌ها به صورت ارائه و دریافت سرویس صورت می‌گیرد و این مولفه‌ها در سه لایه قرار می‌گیرند. لایه‌ی زیرین است که اطلاعات Context را در بر دارد، دسترسی‌ها را تعیین می‌کند و به پایگاه داده نیز متصل است. لایه‌ی میانی به Core Services دسترسی دارد و سرویس‌هایی را برای برنامه‌های کاربردی (Applications) ارائه می‌دهد. و لایه‌ی آخر شامل برنامه‌های کاربردی می‌شود و منطق اصلی بازی در

این لایه قرار دارد. چالش اصلی این معماری مربوط به طراحی پایگاه داده‌ی Context است. این پایگاه داده باید به گونه‌ای طراحی شود که در کاربردهای مختلف قابل گسترش باشد. به این معنی که پایگاه داده‌ی Context قابلیت پشتیبانی از موجودیت‌هایی که به سیستم اضافه می‌شوند و در مرحله‌ی طراحی موجود نبودند را داشته باشد. همچنین قدرت توصیف زبانی که برای Context انتخاب می‌شود اهمیت زیادی دارد.

وجه تمایز دیگر بازی‌های واقعیت فرآگیر، استفاده از ایده‌ها و تکنولوژی‌های مختلف برای تعامل با کاربر است. مقاله‌ی [۴] به معرفی ویژگی‌های یک شیء هوشمند در بازی‌های واقعیت فرآگیر می‌پردازد. مشابه این مفهوم، در کمتر نوع دیگری از بازی مشاهده می‌شود. یکی از نمونه‌های جالب اشیاء هوشمند توسط همین نویسنده معرفی شده است: ساخت یک چوب جادو به وسیله‌ی یک حسگر شتاب، میکروکنترلر و یک تکه چوب.

مقاله‌ی [۵] در جنبه‌ی تعاملی بازی‌های واقعیت فرآگیر به بررسی بازی‌ها بر اساس نوع تعامل پرداخته است. یکی از اصطلاحات جالبی که در این مقاله به آن اشاره شده است، Implicit Player Input است. در واقع اطلاعاتی که از طریق Biosensor ها یا حسگرهای حیاتی متصل به کاربر به جریان بازی وارد می‌شود و بر روند آن تاثیر می‌گذارد، به عنوان ورودی‌های غیرمستقیم کاربر در بازی شناخته می‌شود.

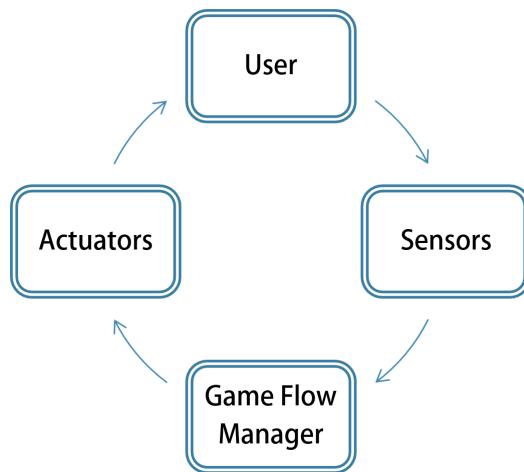
پس از نگاهی مختصر به فعالیت‌هایی که تا کنون در این حوزه انجام شده، می‌توان دریافت که ظرفیت بسیار زیادی برای پیشرفت وجود دارد. چرا که با وجود گذشت حدود دو دهه، هنوز دو جنبه‌ی مربوط به بازی حرف‌های بسیاری برای گفتن دارد. جنبه‌ی نخست استفاده از تمام ظرفیت تکنولوژی در این نوع بازی است. شاید ده سال پیش از این، تکنولوژی به عنوان محدودیتی برای ادامه‌ی فعالیت در این حوزه تلقی می‌شده است؛ اما به نظر می‌رسد اکنون از لحاظ فنی قادر به پاسخگویی نیازهای بازی هستیم. جنبه‌ی دیگر، عرضه‌ی بازی در صنعت است. با توجه به ظرفیت سرگرم‌کنندگی این نوع بازی، به نظر می‌رسد در سال‌های آینده شاهد استقبال مخاطبان و در نتیجه پرداختن صنعتگران به این نوع بازی باشیم.

در این تحقیق، بخشی از جنبه‌ی تعاملی کاربر با بازی که به صورت مستقیم یا غیرمستقیم و با استفاده از حسگرهای انجام می‌شود را مد نظر قرار داده‌ایم. در ادامه ابتدا به معرفی معماری کلی سامانه پرداخته ایم. پس از آن تکنولوژی‌های موجود برای پیاده‌سازی تعامل کاربر با بازی را بررسی کرده‌ایم. و در آخر چگونگی پیاده‌سازی چند ماثول روی بستر انتخاب شده را تشریح کرده‌ایم.

فصل ۳

معماری سامانه

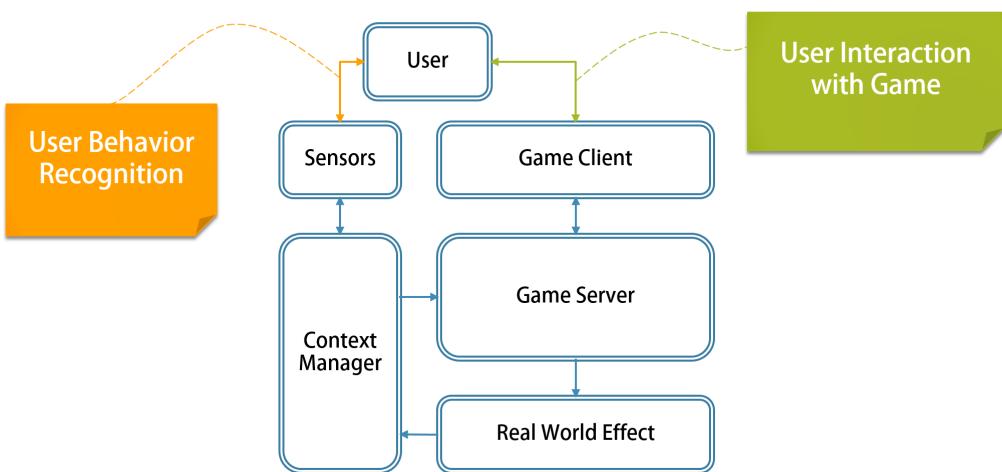
برای طراحی معماری بازی‌های واقعیت فراگیر، ابتدا شکل ساده‌ای از نحوه ارتباط اجزای اصلی را بررسی می‌کنیم:



در این طرح مشخص است که کاربر باید توانایی ارسال ورودی به بازی و دریافت خروجی از بازی را (چه در محیط واقعی و چه در محیط مجازی) داشته باشد. این دو نوع کلی تعامل، با استفاده از حسگرها و عملگرها میسر می‌شوند. از طرف دیگر، مغز متفکر بازی و یا همان Game Flow Manager جاییست که سناریوهای تعریف می‌شوند، مراحل بازی مشخص می‌شود و اطلاعات جایگاه و شرایط کاربر در بازی ثبت می‌شود. بنابراین می‌توان Sensor و Actuator را به عنوان بخش‌های تعاملی بازی و Game Flow Manager را هسته‌ی مرکزی بازی دانست.

در طراحی معماری برای این سامانه باید به نکات مختلفی از جمله مستقل بودن هر مولفه و نحوه ارتباط مولفه‌ها با یکدیگر توجه کرد. به صورتی که داده‌های مرتبط با بازی تنها توسط مولفه‌های قابل دسترسی باشد که به آن داده نیازمند است. و از طرف دیگر از لحاظ معنایی هر مولفه وظیفه‌ی مشخص و مستقلی داشته باشد.

معماری زیر نمونه‌ای از یک معماری برای بازی‌های فراگیر را نشان می‌دهد.



در این معماری تعامل کاربر با بازی را به دو دسته‌ی کلی تقسیم کرده‌ایم. یکی که در واقع دستگاهی است که کاربر به وسیله‌ی آن می‌تواند به بازی ورودی بدهد و خروجی را مشاهده کند. این تعامل ممکن است مربوط به هر یک از دنیای مجازی و واقعی باشد. برای مثال اگر کاربر در خانه نشسته و با رایانه‌ی شخصی بازی می‌کند، رایانه‌ی او همان Game Client است. همچنین اگر در حال بازی در محیط واقعی است و گوشی هوشمندی در دست دارد و با استفاده از آن یک سناریو را برای شروع بازی انتخاب می‌کند، گوشی هوشمند او به عنوان Game Client شناخته می‌شود.

اما حالت دیگری از تعامل کاربر با محیط وجود دارد؛ و آن تشخیص خودکار ویژگی‌ها و شرایط کاربر توسط بازی است. در این حالت کاربر به طور مستقیم با بازی تعامل ندارد، بلکه فعالیت‌های او توسط بازی تشخیص داده می‌شود. این حالت بیشتر در محیط واقعی بازی اتفاق می‌افتد و برای پیاده‌سازی آن لازم است در محیط از حسگرهای مختلف استفاده شود.

طبق تعریف، حسگرهای دستگاه‌هایی هستند که مقادیر یک کمیت فیزیکی را اندازه‌گیری کرده و گزارش می‌کنند. پس استفاده از حسگر به خودی خود شرایط کاربر را به ما اطلاع نمی‌دهد و لازم است روی آن پردازش انجام شود. همچنین گاهی لازم است که با استفاده از اطلاعات به دست آمده از چند

حسگر، نتیجه‌ای حاصل شود. یکی از وظایف Context Manager این است که این داده‌ها را با داده‌های دریافت‌شده از مولفه‌ی Real World Effect تجمعیع کند و فعالیت و شرایط کاربر را تشخیص دهد و به اطلاع مغز بازی یا Game Server برساند.

از طرف دیگر، کاربر باید بتواند بازخوردهای بازی را علاوه بر Game Client در محیط واقعی نیز مشاهده کند. مولفه‌ی Real World Effect دستورالعمل‌های لازم برای تغییر شرایط محیط را از Game Server دریافت می‌کند و با استفاده از Actuator ها این تغییرات را در محیط اعمال می‌کند.

فصل ۴

بررسی تکنولوژی‌های موجود

پس از معرفی مسئله، باید به بررسی راه حل‌های مختلف آن بپردازیم. در این بخش راه‌های موجود را مورد بحث قرار می‌دهیم و دلایل انتخاب‌مان را شرح می‌دهیم.

برای پیاده‌سازی بخش تعامل کاربر با بازی، نیاز به بسترهای داریم که اولاً دارای حسگرهای مورد نیازمان باشد و ثانیاً بتواند اطلاعات خوانده شده توسط حسگر را به بازی ارسال کند. بنابراین از لحاظ فنی ترکیب حسگرها با هر سخت‌افزاری که بتواند اطلاعات را ارسال کند و نوعی شبکه ایجاد کند می‌تواند مورد استفاده قرار بگیرد. اما پاسخ مناسب‌تری وجود دارد. انواع گوشی‌ها و ساعت‌ها و گجت‌های پوشیدنی هوشمندی که امروزه بازار تکنولوژی را پر کرده‌اند، دارای حسگرهای متنوعی هستند و امکان ارسال اطلاعات را نیز پشتیبانی می‌کنند. بنابراین عاقلانه به نظر می‌رسد اگر از اختراع دوباره‌ی چرخ صرف نظر کنیم و بهینه‌ترین راه حل را انتخاب کنیم.

در حال حاضر چند سیستم عامل اصلی در بازار دستگاه‌های پوشیدنی وجود دارد:

Android •

سیستم عامل اندروید بسترهای فیزیکی فراهم کرده است. این سیستم عامل به طور معمول بر گوشی‌های هوشمند نصب می‌شود و در طراحی این سیستم عامل، نکات دستگاه‌های پوشیدنی لحاظ نشده است. اما بسیاری از سازندگان دستگاه‌های پوشیدنی با اعمال تغییراتی در این سیستم عامل آن را برای استفاده در دستگاه‌های پوشیدنی بهینه‌سازی کرده‌اند.

Android Wear •

این سیستم عامل با Android متفاوت است، و برای اجرا به اتصال با یک گوشی اندرویدی با نسخه‌ی ۴.۳ و بالاتر نیاز دارد. نصب یک برنامه که یک نسخه‌ی Wearable در گوشی داشته باشد، باعث می‌شود که آن برنامه در دستگاه Wearable نیز نصب شود. این سیستم عامل تا کنون امکاناتی علاوه بر امکانات سیستم عامل اندروید ارائه نکرده است. بخش‌های زیادی از این سیستم عامل Open Source است اما کد قسمت Google Now که امکان دریافت دستورات صوتی را فراهم می‌کند همچنان در دسترس نیست.

Tizen •

این سیستم عامل که برگرفته از لینوکس است و بخش‌هایی از آن Open Source است، توسط شرکت‌های Intel و Samsung طراحی شده است. این سیستم عامل قابلیت اجرای برنامه‌ها را به صورت مجزا بر روی دستگاه پوشیدنی دارد و نیازی به اتصال به یک گوشی هوشمند ندارد؛ گرچه می‌تواند هماهنگ با گوشی هوشمند نیز عمل کند. برنامه‌های این سیستم عامل با کمک زبان HTML5 تولید می‌شوند. این سیستم عامل همچنین یک نرمافزار به نام Samsung Architecture معرفی می‌کند که بستری برای جمع‌آوری اطلاعات بیومتری کاربران for Multnodal Interactions (مشابه Apple HealthKit و Google Fit) است.

Linux derivative •

سیستم عامل لینوکس در آپدیت‌های جدیدش معماری‌های سخت‌افزاری مختلفی مانند ARM Cortex و MIPS را پشتیبانی می‌کند. به همین دلیل کار با این سیستم عامل بسیار انعطاف‌پذیر است و اکثر سیستم عامل‌های دیگر دستگاه‌های پوشیدنی بر روی هسته‌ی لینوکس طراحی شده‌اند. برخی از دستگاه‌های هوشمند اندروید، iOS و ویندوز با این سیستم عامل سازگار هستند.

Apple Watch OS •

این سیستم عامل تنها با گوشی‌های اپل سازگار است و Open Source نیست.

MediaTek's LinkIt •

این سیستم عامل مخصوص دستگاه‌های پوشیدنی طراحی شده است. بر خلاف سیستم عامل Android Wear که از چیپ‌های گوشی‌های هوشمند استفاده می‌کند، این سیستم عامل بر پایه‌ی چیپ‌های کوچک‌تر، ارزان‌تر و بهینه‌تر کار می‌کند. به همین دلیل عموماً عمر باتری بیشتری دارد.

در این تحقیق، به علت فراگیر بودن و دسترسی بهتر، گوشی‌ها و ساعت‌های هوشمندی که مبتنی بر سیستم عامل اندروید هستند را انتخاب کردیم. این سیستم عامل ارتباط با حسگرها را به خوبی پشتیبانی می‌کند و همچنین انتخاب‌های گسترده‌ای از انواع حسگرها را پیش روی ما قرار می‌دهد. در ادامه به شرح عملکرد حسگرها در سیستم عامل اندروید می‌پردازیم.

حسگرهای اندروید، داده‌ها را به صورت یک دنباله از رویدادها گزارش می‌کنند. هر رویداد شامل موارد زیر است:

- یک handle به حسگری که داده را گزارش داده است.
- زمانی که رویداد اتفاق افتاده است.
- داده‌های مربوط به رویداد.

فهرست حسگرهای فیزیکی موجود در دستگاه‌های اندروید به همراه توضیح مختصری درباره‌ی عملکرد آن در جدول زیر آمده است.

| نام حسگر | وضعیت فعالسازی | توضیحات |
|----------------------------|----------------|--|
| Accelerometer | پیوسته | اندازه‌گیری شتاب در سه محور |
| Ambient temperature | حساس به تغییر | اندازه‌گیری دمای محیط بر حسب سلسیوس |
| Gyroscope | پیوسته | تشخیص جهت دستگاه در سه محور |
| Magnetometers | پیوسته | اندازه‌گیری مقدار میدان مغناطیسی در سه محور |
| Barometer | پیوسته | اندازه‌گیری فشار جو بر حسب هکتوپاسکال |
| Humidity | حساس به تغییر | اندازه‌گیری رطوبت هوای محیط بر حسب درصد |
| Light | حساس به تغییر | اندازه‌گیری روشنایی محیط بر حسب lux |
| Proximity | حساس به تغییر | اندازه‌گیری فاصله‌ی دستگاه تا نزدیک ترین سطح |
| Heart rate | حساس به تغییر | اندازه‌گیری ضربان قلب |

شکل ۱-۴: فهرست حسگرهای اصلی دستگاه‌های اندرویدی.

در دستگاه‌های اندرویدی همچنین تعدادی حسگر ترکیبی تعریف شده که بر پایه‌ی حسگرهایی که در جدول بالا ذکر شد و با به کارگیری ترکیبی از داده‌های آن‌ها عمل می‌کنند. فهرست حسگرهای ترکیبی در ادامه آمده است.

به جز موارد ذکر شده در بالا، دسته‌ی دیگری از حسگرهای اندرویدی وجود دارند که به تعامل

| نام حسگر | وضعیت فعالسازی | توضیحات |
|---|---------------------|---|
| Linear Accelerometer | پیوسته | اندازه‌گیری شتاب خطی (بدون دخیل کردن جاذبه) |
| Signifant Motion | نیازمند فعالسازی | اندازه‌گیری تغییرات قابل توجه در موقعیت کاربر (برای مثال راه رفتن یا نشستن در خودروی در حال حرکت) |
| Step Detector or Step Counter | حساس به گام برداشتن | شمارش تعداد گام‌ها |
| Tilt Detector | حساس به تغییر | فعالسازی هنگام تغییر بیش از ۳۵ درجه در کمتر از ۲ ثانیه |
| Rotation Vector | پیوسته | تشخیص جهت دستگاه نسبت به شمال شرق (یعنی x به سمت شرق و موازی با زمین، y به سمت شمال و موازی با زمین، z به سمت آسمان و عمود بر زمین) |
| Game Rotation Vector | پیوسته | مشابه Rotation Vector با این تفاوت که محور y می‌تواند در جهت دیگری قرار داشته باشد. |
| Gravity or Geomagnetic Rotation Vector | پیوسته | تشخیص جهت و اندازه‌ی جاذبه در مختصات دستگاه |

شکل ۴-۲: فهرست حسگرهای ترکیبی دستگاه‌های اندرویدی.

مستقیم کاربر با گوشی مرتبط هستند. فهرست حسگرهای تعاملی نیز در جدولی ارائه شده است.

| نام حسگر | توضیحات |
|------------------------|--|
| Wake Up Gesture | عملکرد مشابه با دکمه‌ی power گوشی (روشن شدن صفحه‌ی گوشی) |
| Pick Up Gesture | فعالسازی هنگام برداشتن گوشی (از روی میز، درون جیب،...) |
| Glance Gesture | روشن شدن موقت صفحه‌ی گوشی |

شکل ۴-۳: فهرست حسگرهای تعاملی دستگاه‌های اندرویدی.

فصل ۵

ماژول‌ها

۱-۵ مژول تشخیص فعالیت‌های کاربر

۱-۱-۵ معرفی مژول

هدف این مژول تشخیص حالت فیزیکی و فعالیت‌های کاربر، و ذخیره‌ی آن‌ها در یک پایگاه داده است. این برنامه از سرویس Google API Client استفاده می‌کند. برای استفاده از این سرویس، باید کتابخانه Google Play Services را در برنامه به کار ببریم. استفاده از این سرویس نیازی به ارتباط با اینترنت ندارد.

این برنامه برای گوشی‌های اندرویدی با ورژن اندروید (API 16) 4.1 و بالاتر نوشته شده است.

فهرست فعالیت‌های احتمالی کاربر که توسط Google API Client تشخیص داده می‌شود:

1. In Vehicle

2. On Bicycle

3. Walking Running

4. On Foot

5. Still

6. Tilting

نکات دیگری که باید در مورد این اپلیکیشن و سرویس Google API در نظر گرفته شود:

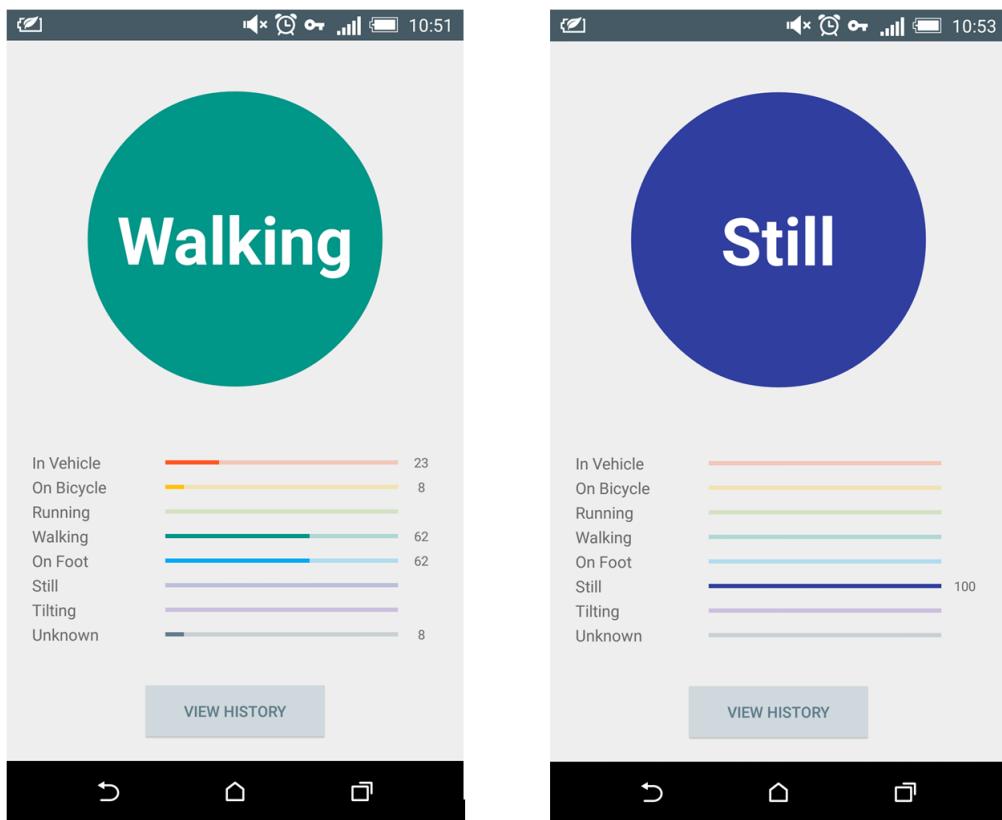
- اپلیکیشن هر دو ثانیه یک بار به سرویس درخواست داده و نتایج آن را در صفحه نمایش می‌دهد.
- این سرویس با استفاده از داده‌های حسگرهای مختلف گوشی، فعالیت‌ها را تشخیص می‌دهد. ممکن است زمان لازم برای تشخیص فعالیت‌های احتمالی، بیشتر از زمانی باشد که ما درخواست داده‌ایم. زیرا این سرویس برای تشخیص فعالیت، تنها از داده‌های همان لحظه استفاده نمی‌کند و به دنباله‌ای از داده‌ها احتیاج دارد. در صورتی که بعد از گذشت دو ثانیه، اطلاعات روی صفحه به روز رسانی نشود، به این معناست که سرویس داده‌ی جدیدی برای ارائه ندارد. به محض آن که اطلاعات جدیدی در مورد فعالیت کاربر وجود داشته باشد، صفحه به روز رسانی می‌شود.
- این سرویس از حسگرهایی استفاده می‌کند که باتری زیادی مصرف نمی‌کنند.
- در دستگاه‌هایی که دارای حسگر TYPE SIGNIFICANT MOTION هستند، اگر حالت دستگاه برای مدت زیادی Still یا ثابت باشد، سرویس تشخیص فعالیت را متوقف می‌کند. علت این امر، صرفه جویی در مصرف باتری است.
- استفاده از این سرویس به permission.ACTIVITY RECOGNITION احتیاج دارد.
- این اپلیکیشن داده‌های موجود در پایگاه داده را در فهرست‌هایی به تفکیک روز به کاربر نمایش می‌دهد.

۲-۱-۵ نمونه‌ای از کارکرد برنامه

هنگامی که کاربر برنامه را باز می‌کند، تا زمانی که سرویس Google API اطلاعاتی را به برنامه ارسال کند، علامت Loading روی صفحه نشان داده می‌شود. با دریافت اولین اطلاعات توسط برنامه، دو بخش زیر در صفحه قابل مشاهده است:

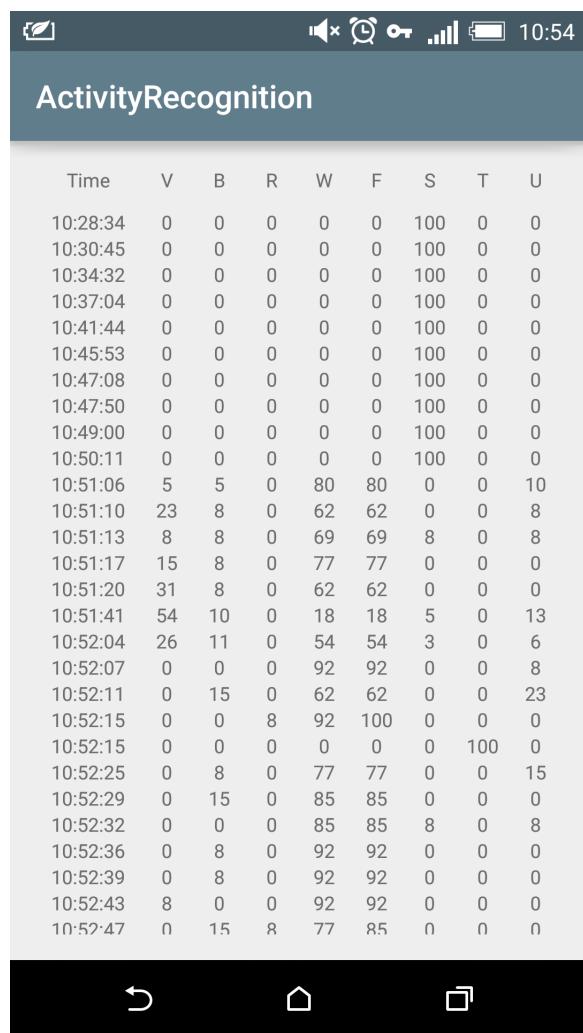
- عنوان وضعیت احتمالی کاربر: فعالیتی که با توجه به داده‌های دریافتی از سرویس گوگل، بیشترین احتمال و مقدار قطعیت را دارد.

– میزان احتمال هریک از فعالیت‌های ممکن کاربر، که توسط یک Progress Bar نمایش داده می‌شود. احتمال هر فعالیت عددی بین صفر تا صد است. تصویر ۱-۵ صفحه‌ی برنامه را در دو حالت مختلف نشان می‌دهند.



شکل ۱-۵: تصاویری از صفحه‌ی برنامه‌ی تشخیص فعالیت‌های کاربر در دو حالت ثابت و در حال راه رفتن.

تصویر ۲-۵ داده‌های ثبت شده در پایگاه داده را نشان می‌دهد. هر سطر، نشان‌دهنده‌ی میزان هریک از انواع فعالیت‌های کاربر در یک لحظه از زمان است. (اولین سطر به عنوان راهنمای قرار داده شده است و شامل حرف اول نام هر فعالیت است).

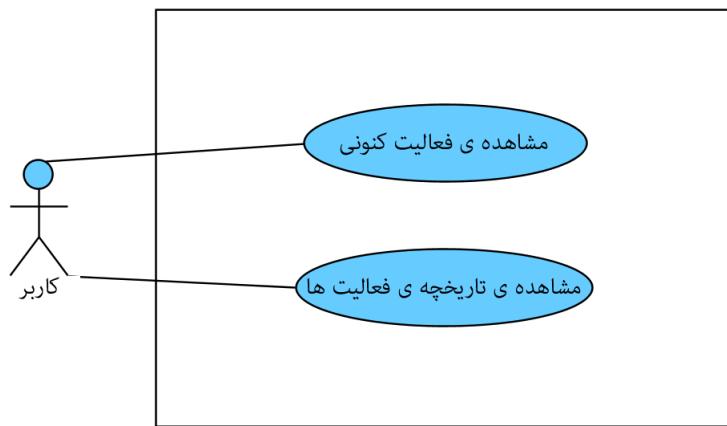


| Time | V | B | R | W | F | S | T | U |
|----------|----|----|---|----|-----|-----|-----|----|
| 10:28:34 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:30:45 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:34:32 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:37:04 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:41:44 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:45:53 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:47:08 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:47:50 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:49:00 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:50:11 | 0 | 0 | 0 | 0 | 0 | 100 | 0 | 0 |
| 10:51:06 | 5 | 5 | 0 | 80 | 80 | 0 | 0 | 10 |
| 10:51:10 | 23 | 8 | 0 | 62 | 62 | 0 | 0 | 8 |
| 10:51:13 | 8 | 8 | 0 | 69 | 69 | 8 | 0 | 8 |
| 10:51:17 | 15 | 8 | 0 | 77 | 77 | 0 | 0 | 0 |
| 10:51:20 | 31 | 8 | 0 | 62 | 62 | 0 | 0 | 0 |
| 10:51:41 | 54 | 10 | 0 | 18 | 18 | 5 | 0 | 13 |
| 10:52:04 | 26 | 11 | 0 | 54 | 54 | 3 | 0 | 6 |
| 10:52:07 | 0 | 0 | 0 | 92 | 92 | 0 | 0 | 8 |
| 10:52:11 | 0 | 15 | 0 | 62 | 62 | 0 | 0 | 23 |
| 10:52:15 | 0 | 0 | 8 | 92 | 100 | 0 | 0 | 0 |
| 10:52:15 | 0 | 0 | 0 | 0 | 0 | 0 | 100 | 0 |
| 10:52:25 | 0 | 8 | 0 | 77 | 77 | 0 | 0 | 15 |
| 10:52:29 | 0 | 15 | 0 | 85 | 85 | 0 | 0 | 0 |
| 10:52:32 | 0 | 0 | 0 | 85 | 85 | 8 | 0 | 8 |
| 10:52:36 | 0 | 8 | 0 | 92 | 92 | 0 | 0 | 0 |
| 10:52:39 | 0 | 8 | 0 | 92 | 92 | 0 | 0 | 0 |
| 10:52:43 | 8 | 0 | 0 | 92 | 92 | 0 | 0 | 0 |
| 10:52:47 | 0 | 15 | 8 | 77 | 85 | 0 | 0 | 0 |

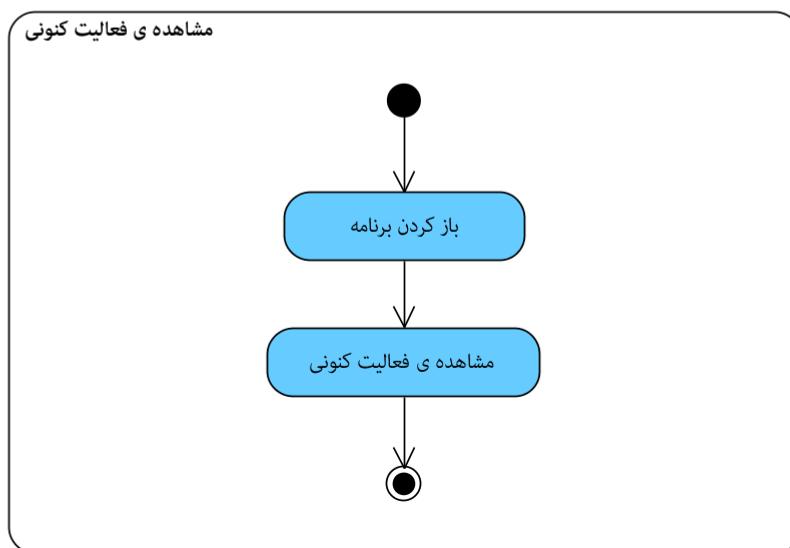
شکل ۵-۲: تصویری از پایگاه داده‌ی برنامه‌ی تشخیص فعالیت‌های کاربر.

۳-۱-۵ نمودارهای تحلیل و طراحی

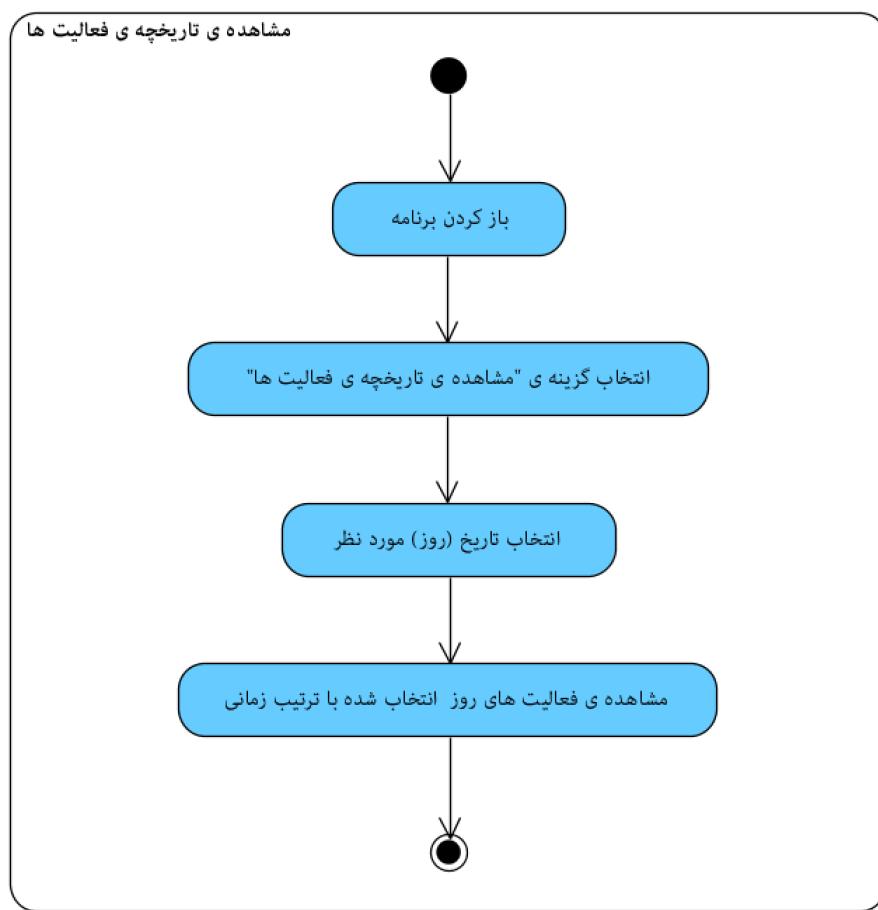
همان‌طور که در تصویر ۳-۵ آمده است، نمودار مورد کاربرد این برنامه شامل دو مورد مشاهده‌ی فعالیت کنونی و مشاهده‌ی تاریخچه‌ی فعالیت‌ها است. نمودار فعالیت برای هر یک از این دو مورد کاربر در تصاویر ۴-۵ و ۵-۵ آمده است.



شکل ۳-۵: نمودار مورد کاربرد برنامه‌ی تشخیص فعالیت‌های کاربر.

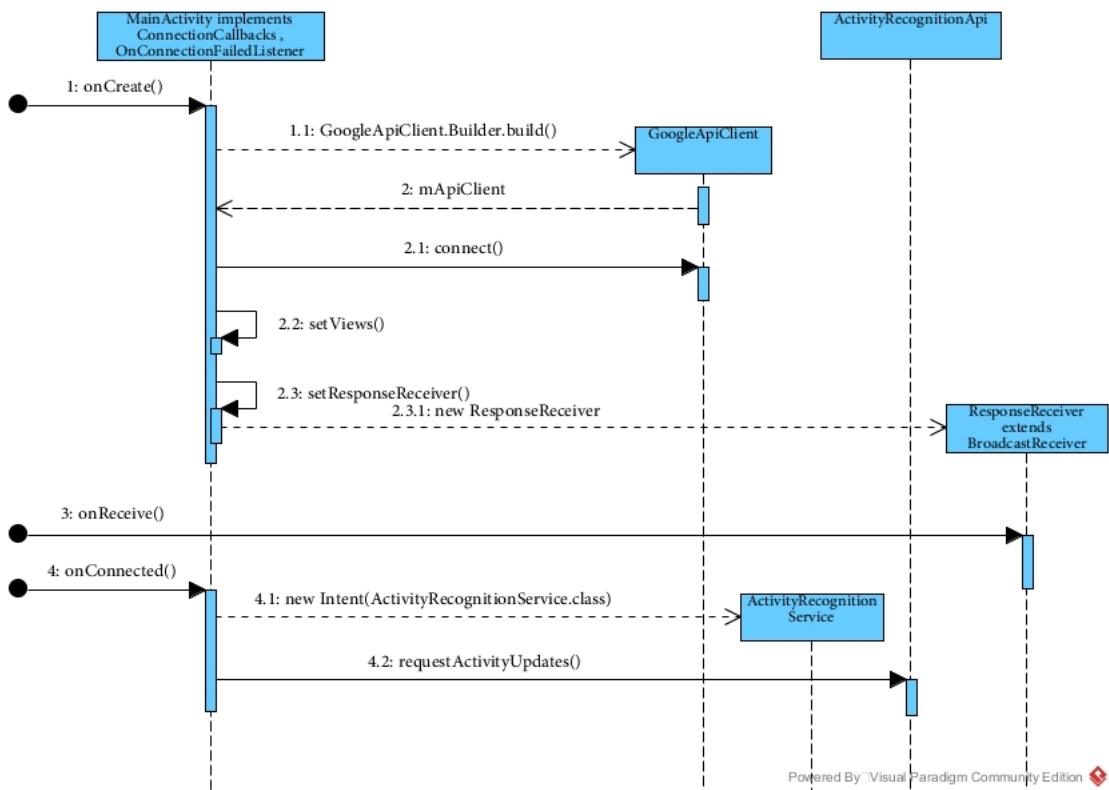


شکل ۴-۵: نمودار فعالیت برای مورد کاربرد مشاهده‌ی فعالیت کنونی کاربر.

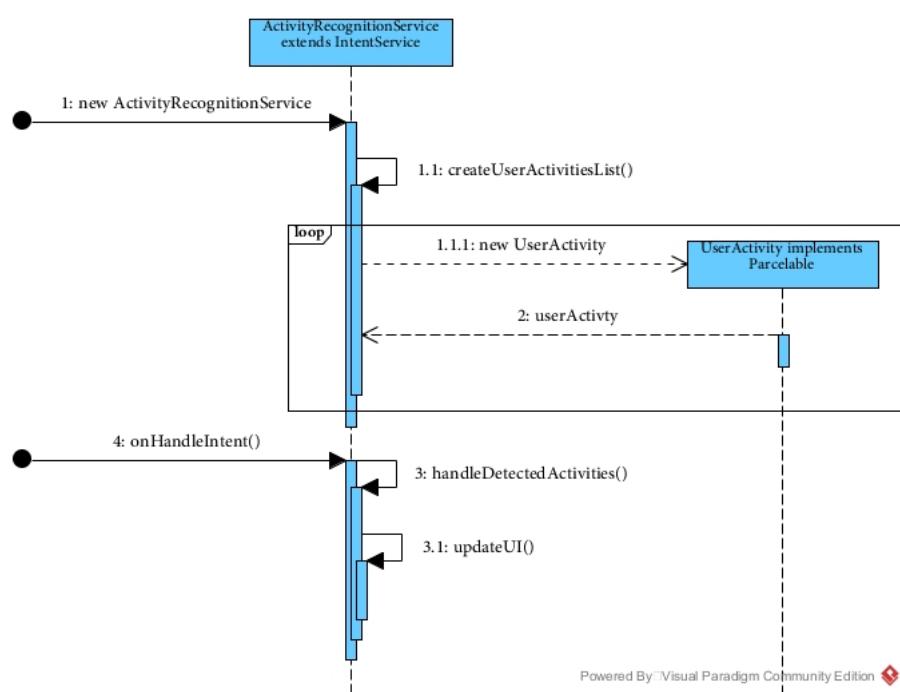


شکل ۵-۵: نمودار فعالیت برای مورد کاربرد مشاهده‌ی تاریخچه‌ی فعالیت‌های کاربر.

این برنامه شامل دو کلاس اصلی `ActivityRecognitionService` و `MainActivity` است. در ادامه نمودار توالی این دو کلاس که نشان‌گر نحوه تعامل قسمت‌های مختلف برنامه است، آورده شده است. همچنین در قسمت شرح پیاده‌سازی به جزئیات بیشتری در مورد این دو کلاس پرداخته‌ایم.



شکل ۵-۶: نمودار توالی کلاس `MainActivity`



شکل ۵-۷: نمودار توالی کلاس ActivityRecognitionService

۴-۱-۵ پیاده‌سازی

برای این که بتوانیم اطلاعات فعالیت‌های کاربر را در بازه‌های زمانی مشخص (حتی زمانی که کاربر برنامه را باز نکرده است)، داشته باشیم، نیاز داریم که برنامه همواره در پس‌زمینه در حال اجرا باشد. به طور کلی در اندروید، برای این که یک عمل به صورت پیوسته در پس‌زمینه اجرا شود، نیاز به Service داریم. Service‌ها در اندروید، UI ندارند و از چرخه‌ی Activity تبعیت نمی‌کنند. یک Service به طور عادی در Main Thread اجرا می‌شود. یعنی اگر Task‌های طولانی داشته باشیم، تا پایان اجرای Service صفحه Freeze می‌شود. برای این که در یک Thread جدا اجرا شود، باید یک Working Thread در Service تعریف شود. اما راه حل ساده‌تری هم وجود دارد. کلاس IntentService، که Service را Extend می‌کند، در یک Working Thread اجرا می‌شود و به خودی خود ارتباطی با Main Thread ندارد. پس خواندن اطلاعات فعالیت کاربر در یک IntentService انتخاب خوبی به نظر می‌رسد. در این برنامه، دو کلاس اصلی وجود دارد:

۱. کلاس MainActivity: وظیفه‌ی کلی این کلاس، اتصال به Google API Client، آغاز یک Service در background، دریافت اطلاعات از Service، و نمایش اطلاعات دریافت شده در UI است.

۲. کلاس ActivityRecognitionService: این کلاس که توسط MainActivity ایجاد می‌شود، همواره در حال اجراست و اطلاعات را از Google API Client دریافت می‌کند، در پایگاه داده ثبت می‌کند، و به MainActivity ارسال می‌کند.

ارتباط بین این دو کلاس، با استفاده از Broadcast Intent کردن یک Broadcast می‌شود. به این معنی که هنگامی که اطلاعات جدیدی در Service وجود دارد، یک Intent Service می‌سازد و با یک Action خاص آن را Broadcast می‌کند. از طرف دیگر، کلاس MainThread نیز همواره منتظر دریافت یک Intent با همان Action خاص است. پس اطلاعات از سمت Service ارسال شده و در UI دریافت می‌شود.

نکته‌ی دیگر، نوع اطلاعاتی است که بین Service و UI جابه‌جا می‌شود. کلاس UserActivity یک کلاس data است که سه مشخصه دارد. نام فعالیت، رنگ تخصیص داده شده به آن فعالیت، و میزان احتمال انجام فعالیت توسط کاربر (عددی بین صفر تا صد). در هر بار دریافت اطلاعات از

MainActivity، اطلاعات یک آرایه در Service به روز رسانی شده و به Google API ارسال می‌شود.

کلاس MainActivity به صورت زیر تعریف می‌شود:

```
public class MainActivity
    extends AppCompatActivity
    implements GoogleApiClient.ConnectionCallbacks,
    GoogleApiClient.OnConnectionFailedListener {
    ...
}
```

این کلاس سه بخش کلی دارد:

- متد () که در آغاز برنامه صدای زده می‌شود و برنامه را initiate می‌کند.
- متدهایی که در کلاس‌های implement شده وجود دارند و باید override شوند.
- تعریف کلاس داخلی (inner class) به نام ResponseReceiver که برای دریافت اطلاعات از ActivityRecognitionService به کار می‌رود.

در ادامه به تشریح هریک از این سه بخش می‌پردازیم.

در تابع () onCreate، سه کار اصلی انجام می‌شود:

۱. دستیابی به عناصر UI

۲. اتصال به Google API Client

```
mApiClient = new GoogleApiClient.Builder(this)
    .addApi(ActivityRecognition.API)
    .addConnectionCallbacks(this)
    .addOnConnectionFailedListener(this)
    .build();

mApiClient.connect();
```

هنگام تعریف API Client Google API را مشخص کنیم:

همچنین، برای هر API Google، باید دو مورد زیر را تعیین کنیم: Connection Callback که یک interface است و دو متدهای onConnected() و onConnectionSuspended() را دارد. این دو callback به ترتیب هنگامی که اتصال برقرار شد و یا اتصال در حالت تعليق قرار گرفت صدا زده می‌شوند. در این برنامه، MainActivity این interface را implement کند. پس در کد بالا، this را به عنوان اشاره‌گری به MainActivity پاس داده‌ایم. و همچنین onConnectionFailedListener که مانند مورد قبل یک interface است و یک متدهای onConnectionFailed() را دارد. این متدهای زمانی صدا زده می‌شود که در اتصال به API Google مشکلی ایجاد شود. Google API Client این interface را نیز implement کند پس this را به MainActivity پاس می‌دهیم.

۳. تعریف یک BroadcastReceiver برای دریافت اطلاعات ارسالی از ActivityRecognitionService کلاس ResponseReceiver برای دریافت Intent از ActivityRecognitionService تعیین شده است. این کلاس باید در متد onCreate() نمونه‌سازی شده و به شکل زیر مورد استفاده قرار بگیرد:

```
IntentFilter filter = new IntentFilter(ResponseReceiver.ACTION_RESPONSE);
filter.addCategory(Intent.CATEGORY_DEFAULT);
ResponseReceiver receiver = new ResponseReceiver(this);
registerReceiver(receiver, filter);
```

بخش بعدی، مربوط به متدهای override شده است.

- متدهای `onConnected` و `onConnectionSuspended` از کلاس `ConnectionCallback` هستند که در کتابخانه `Google API Client` با `MainActivity` ارتباط برقرار شده باشد.
- کلاس `ActivityRecognition` که در کتابخانه `Google API Client` قرار دارد، متدهای `requestActivityUpdates` و `cancelActivityUpdates` دارد که یک `PendingIntent` به عنوان ورودی دریافت می‌کند. تفاوت `PendingIntent` با `Intent` این است که `PendingIntent` از `Permission` های برنامه می‌فرستد. برای مثال، در اینجا `PendingIntent` در `MainActivity` ساخته شده پس از `Request` `Permission` برنامه می‌استفاده می‌کند. ورودی دیگر این متدها، زمان بین `onConnected` و `onConnectionSuspended` است و در اینجا این زمان را ۲ ثانیه تعریف کرده‌ایم. برای ساخت `PendingIntent` می‌توانیم

از یک Service استفاده کنیم و در اینجا از کلاس ActivityRecognitionService استفاده کردایم:

```
Intent intent = new Intent(this, ActivityRecognitionService.class);
PendingIntent pendingIntent =
PendingIntent.getService(this, 0, intent,
PendingIntent.FLAG_UPDATE_CURRENT);
ActivityRecognition.ActivityRecognitionApi.requestActivityUpdates(mApiClient,
2000, pendingIntent);
```

– متدهای onConnectionFailed و onConnectionSuspended از کلاس ConnectopnCallback

از کلاس OnConnectionFailedListener:

در این دو متدهای کافیست که بار دیگر برای برقراری ارتباط با Google API Client، تابع connect() را فراخوانی کنیم.

بخش آخر در MainActivity مربوط به تعریف کلاس ResponseActivity است. همان‌طور که در شرح کلی برنامه توضیح داده شد، برای برقراری ارتباط بین MainActivity و ActivityRecognitionService به یک Intent احتیاج داریم که از طرف سرویس ارسال شود و در UI دریافت شود. به این منظور، کلاس داخلی ResponseReceiver را Extend کرده‌ایم:

```
public class ResponseReceiver extends BroadcastReceiver {
    ...
}
```

گفتیم که برای متمایز کردن این Intent از Intent های برنامه‌های دیگر، از یک Action خاص ResponseReceiver می‌کنیم. این Action به صورت یک String static در کلاس ResponseReceiver تعریف شده و توسط هر دو طرف ارسال‌کننده و گیرنده استفاده می‌شود:

```
public static final String ACTION_RESPONSE =
"com.sharif.hcilab.activityrecognition.ACTION_RESPONSE";
```

کلاس BroadcastReceiver یک متد onReceive() دارد که باید توسط کلاس‌های فرزند، override شود. این متد هنگامی صدازده می‌شود که یک Intent توسط یک instant از این کلاس دریافت شود. اطلاعات ارسالی از سرویس، به صورت یک آرایه از کلاس UserActivity است. این آرایه با نام ”USER ACTIVITIES“ به Intent ارسالی attach شده است. به کمک دستور زیر، این آرایه را در متد onReceive() دریافت می‌کنیم:

```
userActivities = intent.getParcelableArrayListExtra("USER_ACTIVITIES");
```

حال می‌توانیم اطلاعات این آرایه را به هر صورتی که خواستیم در UI نمایش بدهیم. در این برنامه با استفاده از Recycler View و ProgressBar احتمال هر

همچنین title فعالیتی که بیشترین احتمال را دارد در بالای صفحه نشان می‌دهیم.

کلاس بعدی کلاس ActivityRecognitionService است. هدف این کلاس، دریافت اطلاعات از Thread و ارسال آن به MainActivity Google API Client extend IntentService و در حال اجرا باشد، پس کلاس MainThread را جدا از background می‌کند.

در constructor این کلاس، آرایه‌ی کلاس‌های فعالیت کاربر را که از جنس کلاس UserActivity است intantiate می‌کنیم. (همان‌گونه که گفته شد، کلاس UserActivity به منظور نگهداری مشخصات مربوط به فعالیت‌های کاربر ایجاد شده است). این آرایه با هر بار دریافت اطلاعات به روز رسانی شده و به MainActivity ارسال می‌شود.

```
public void createUserActivitiesList() {
    userActivities = new ArrayList<>();
    userActivities.add(new UserActivity("In Vehicle", R.color.orange));
    userActivities.add(new UserActivity("On Bicycle", R.color.yellow));
    userActivities.add(new UserActivity("Running", R.color.green));
    userActivities.add(new UserActivity("Walking", R.color.teal));
    userActivities.add(new UserActivity("On Foot", R.color.blue));
    userActivities.add(new UserActivity("Still", R.color.indigo));
    userActivities.add(new UserActivity("Tilting", R.color.purple));
    userActivities.add(new UserActivity("Unknown", R.color.grey));
}
```

}

با extend IntentService کردن متد onHandleIntent()، امکان فراهم می‌شود؛ این متد یک Intent را به عنوان آرگومان دریافت می‌کند که در اینجا اطلاعات فعالیت‌های کاربر را به همراه دارد:

```
@Override
protected void onHandleIntent(Intent intent) {
    if (ActivityRecognitionResult.hasResult(intent)) {
        ActivityRecognitionResult result =
            ActivityRecognitionResult.extractResult(intent);
        handleDetectedActivities(result.getProbableActivities());
    }
}
```

حال در تابع handleDetectedActivities() میزان احتمال یا confidence هر فعالیت را در آرایه به روز رسانی می‌کنیم:

```
switch (activity.getType()) {
    case DetectedActivity.IN_VEHICLE: {
        userActivities.get(0).setConfidence(activity.getConfidence());
        break;
    }
    ...
}
```

و در نهایت این آرایه را به MainActivity ارسال می‌کنیم:

```
Intent broadcastIntent = new Intent();
broadcastIntent.setAction(MainActivity.ResponseReceiver.ACTION_RESPONSE);
broadcastIntent.addCategory(Intent.CATEGORY_DEFAULT);
broadcastIntent.putParcelableArrayListExtra("USER_ACTIVITIES",
    userActivities);
```

```
sendBroadcast(broadcastIntent);
```

به این صورت، اطلاعات از سرویس به UI منتقل شده و نمایش داده می‌شود. برای پیاده‌سازی پایگاه داده، یک جدول برای ثبت داده‌های فعالیت کاربر در پایگاه داده لازم داریم. این جدول شامل تاریخ، زمان، و میزان هر یک از فعالیت‌های هشت‌گانه‌ی کاربر است. در این برنامه از پایگاه داده‌ی خود اندروید یعنی SQLite استفاده کرده‌ایم. برای کار با این پایگاه داده، کلاس‌ی extend SQLiteOpenHelper را می‌کنیم:

```
public class DatabaseHandler extends SQLiteOpenHelper
```

تعریف جدول:

```
@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {

    String CREATE_USER_ACTIVITY_TABLE = "CREATE TABLE " + USER_ACTIVITY_TABLE
        + "("
        + KEY_ID + " INTEGER PRIMARY KEY,"
        + KEY_DATE + " DATE,"
        + KEY_TIME + " TIME,"
        + KEY_IN_VEHICLE + " INTEGER,"
        + KEY_ON_BICYCLE + " INTEGER,"
        + KEY_RUNNING + " INTEGER,"
        + KEY_WALKING + " INTEGER,"
        + KEY_ON_FOOT + " INTEGER,"
        + KEY_STILL + " INTEGER,"
        + KEY_TILTING + " INTEGER,"
        + KEY_UNKNOWN + " INTEGER"
        + ")";
    sqLiteDatabase.execSQL(CREATE_USER_ACTIVITY_TABLE);
}
```

}

افزودن داده به جدول:

```
ContentValues values = new ContentValues();
values.put(KEY_IN_VEHICLE, userActivities.get(0).getConfidence());
...
// Inserting Row
sqLiteDatabase.insert(USER_ACTIVITY_TABLE, null, values);
```

در این برنامه، داده‌ها را به تفکیک روز به کاربر نشان می‌دهیم. پس نیاز داریم که فهرست روزها را از پایگاه داده دریافت کنیم. متد زیر این کار را انجام می‌دهد:

```
Cursor cursor = sqLiteDatabase.query(true, USER_ACTIVITY_TABLE,
    new String[]{KEY_DATE}, null, null, KEY_DATE, null, null);
if (cursor.moveToFirst()) {
    do {
        dateArrayList.add(cursor.getString(0));
    } while (cursor.moveToNext());
}
```

همچنین نیاز داریم که داده‌های مربوط به هر روز را، با دریافت تاریخ آن روز، از پایگاه داده بخوانیم. کوئری زیر این کار را انجام می‌دهد:

```
String selectQuery = "SELECT * FROM " + USER_ACTIVITY_TABLE
    + " WHERE " + KEY_DATE + " = '" + date + "'";
```

به این ترتیب، امکان ثبت و دریافت داده‌ها در پایگاه داده فراهم می‌شود.

۲-۵ مژول خواندن تگ NFC

۱-۲-۵ معرفی مژول

این مژول برای گوشی‌های اندرویدی با ورژن اندروید ۴.۰ (API 14) و بالاتر نوشته شده است. هدف این اپلیکیشن، خواندن اطلاعات آن دسته از Tag های NFC است که به فرمت NDEF ثبت شده‌اند و از جنس متن هستند. در این اپلیکیشن همچنین وضعیت NFC (فعال/غیرفعال بودن) نشان داده می‌شود و دسترسی به تنظیمات NFC وجود دارد.

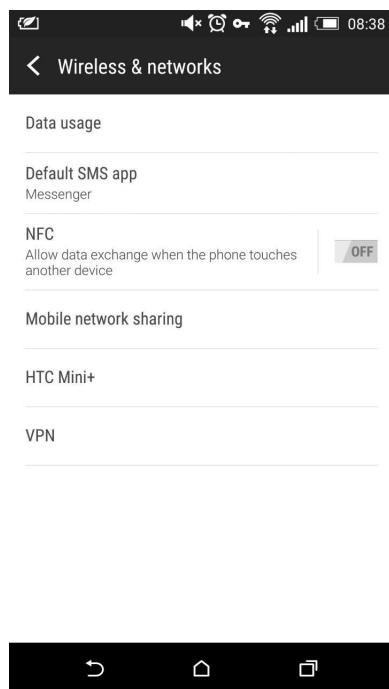
۲-۲-۵ نمونه‌ای از کارکرد برنامه

هنگامی که کاربر برنامه را باز می‌کند با صفحه‌ای مانند شکل ۸-۵ مواجه می‌شود. در این صفحه وضعیت NFC نمایش داده می‌شود و دکمه‌ای برای باز کردن تنظیمات NFC موجود است. همچنین در نوار سبز رنگ مشاهده می‌کنیم که هنوز هیچ تگی در اطراف دستگاه شناسایی نشده است.



شکل ۸-۵: تصویری از صفحه‌ی نخست برنامه‌ی خواندن تگ NFC.

با زدن دکمه‌ی تنظیمات NFC، صفحه‌ی مربوط به تنظیمات گوشی و قسمتی که تنظیمات NFC وجود دارد باز شده و کاربر می‌تواند NFC را فعال/غیرفعال کند. (تصویر ۹-۵) لازم به ذکر است که در سیستم عامل اندروید اجازه‌ی تغییر در تنظیمات گوشی به اپلیکیشن‌ها داده نمی‌شود و برای تغییر آن‌ها کاربر باید حتماً به صفحه‌ی تنظیمات برود.



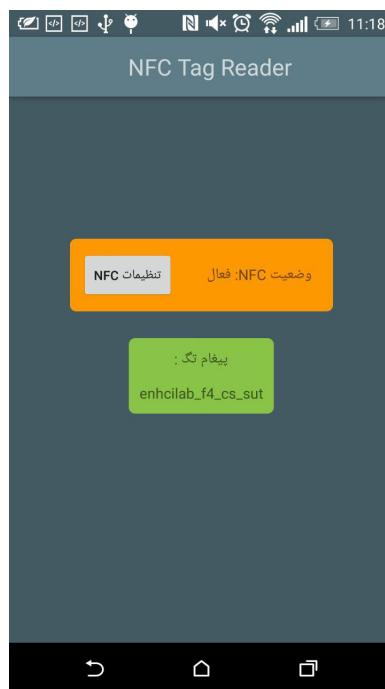
شکل ۹-۵: تصویری از صفحه‌ی تنظیمات NFC.

پس از فعال کردن NFC، با زدن دکمه‌ی back، دوباره صفحه‌ی برنامه نمایش داده می‌شود و مشاهده می‌کنیم که وضعیت NFC تغییر پیدا کرده و فعال شده است. (تصویر ۱۰-۵)



شکل ۱۰-۵: تصویری از صفحه‌ی نخست برنامه پس از فعال کردن NFC.

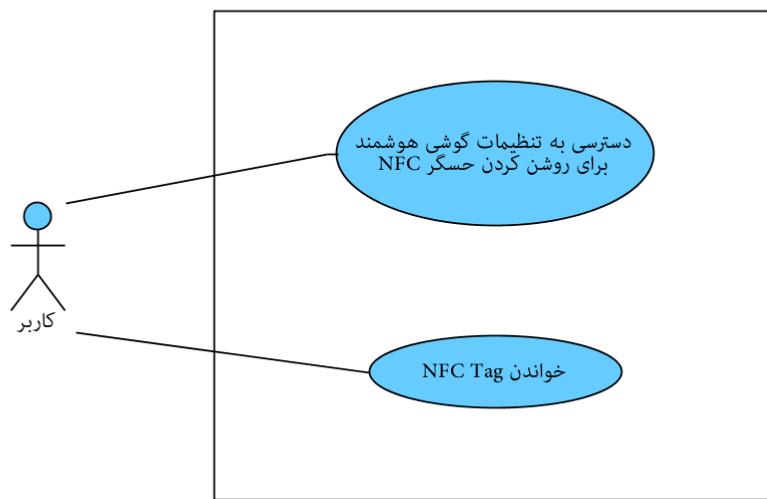
حال گوشی را با یک NFC Tag امتحان می‌کنیم. برای خواندن تگ، گوشی را روی تگ و با فاصله‌ی خیلی کم نگه می‌داریم. پس از خواندن اطلاعات توسط گوشی، اطلاعات ثبت شده در تگ، در نوار سبز رنگ ظاهر می‌شود. (تصویر ۱۱-۵)



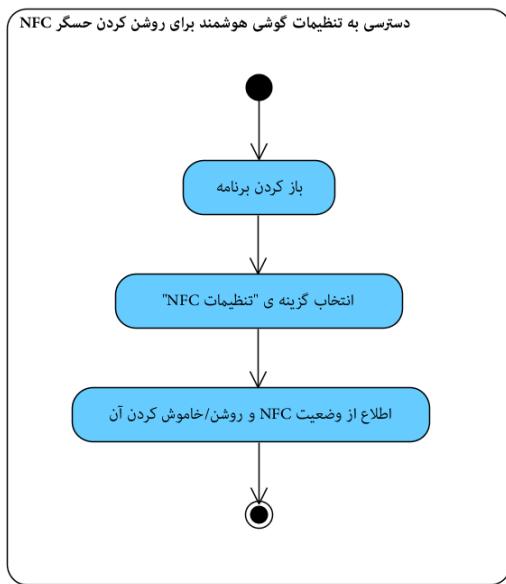
شکل ۱۱-۵: تصویری از نمایش اطلاعات تگ در برنامه.

۳-۲-۵ نمودارهای تحلیل و طراحی

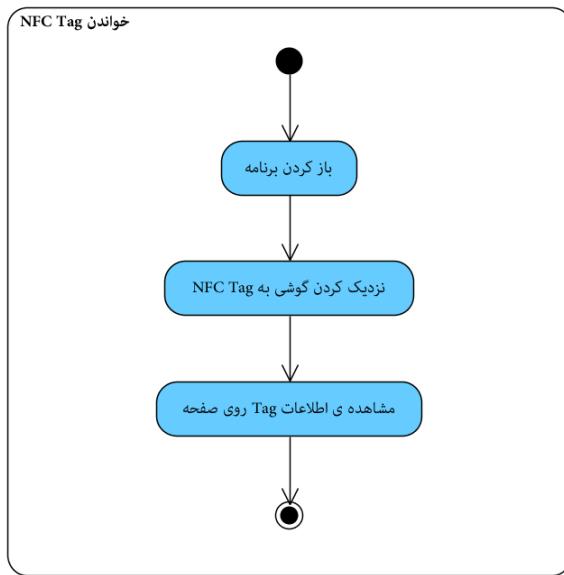
نمودار مورد کاربرد برای این برنامه شامل دو مورد دسترسی به تنظیمات NFC و خواندن اطلاعات تگ است. (تصویر ۱۲-۵)



شکل ۱۲-۵ : نمودار موارد کاربرد برنامه‌ی NFC Tag Reader. نمودار فعالیت برای دو مورد کاربرد این برنامه در تصاویر ۱۳-۵ و ۱۴-۵ آمده است.

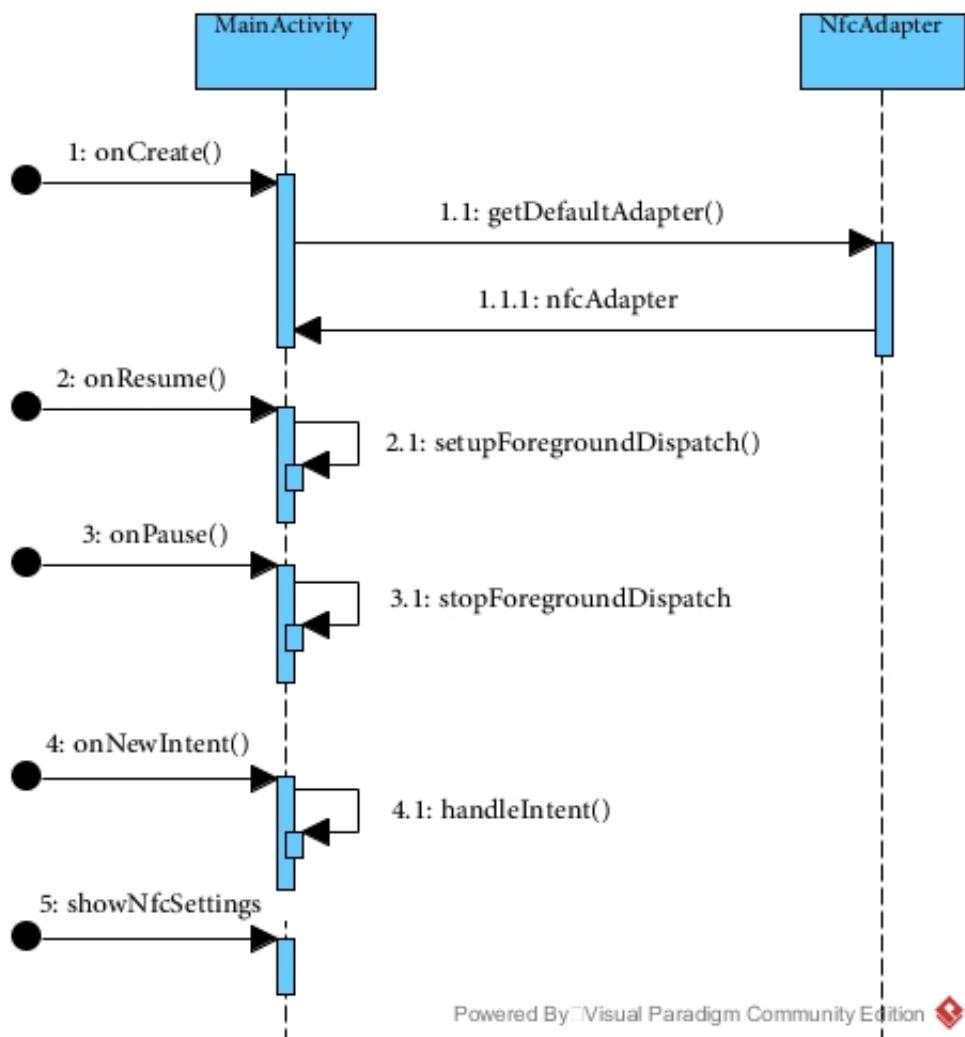


شکل ۱۳-۵: نمودار فعالیت برای مورد کاربرد دسترسی به تنظیمات NFC.



شکل ۱۴-۵: نمودار فعالیت برای مورد کاربرد خواندن تگ.

نمودار توالی کلاس اصلی این برنامه یعنی MainActivity در تصویر ۱۵-۵ نشان داده شده است.



شکل ۱۵-۵: نمودار توالی کلاس NFC Tag Reader از برنامه‌ی MainActivity

۴-۲-۵ پیاده‌سازی

ابتدا باید در متد onCreate از کلاس اصلی، Adapter مربوط به NFC را بگیریم:

```
mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
```

اگر مقدار این Adapter برابر با null باشد، یه این معنی است که این دستگاه تکنولوژی NFC را پشتیبانی نمی‌کند. بنابراین برنامه "پیغام این دستگاه NFC را پشتیبانی نمی‌کند" را به صورت Toast نشان می‌دهد و بسته می‌شود:

```
if (mNfcAdapter == null) {
    Toast.makeText(this, R.string.nfc_not_supported, Toast.LENGTH_LONG).show();
    finish();
    return;
}
```

برای کنترل روش‌ن بودن NFC، در تابع onResume برسی می‌کنیم که آیا Adapter فعال است یا خیر، و وضعیت را در نوار نارنجی رنگ نشان می‌دهیم:

```
if (mNfcAdapter.isEnabled()) {
    nfcStatusTextView.setText(R.string.nfc_enabled);
} else {
    nfcStatusTextView.setText(R.string.nfc_disabled);
}
```

برای تغییر وضعیت NFC، باید تابعی تعریف کنیم که صفحه‌ی تنظیمات گوشی را نشان دهد. نحوه‌ی دسترسی به این صفحه از ورژن ۱۶ SDK تغییر کرده است. پس ابتدا ورژن دستگاه را بررسی کرده و سپس صفحه‌ی مربوطه را باز می‌کنیم:

```
public void showNfcSetting(View view) {
    if (android.os.Build.VERSION.SDK_INT >= 16) {
        startActivity(new Intent(android.provider.Settings.ACTION_NFC_SETTINGS));
    }
}
```

```

} else {
    startActivity(
        new Intent(android.provider.Settings.ACTION_WIRELESS_SETTINGS));
}
}

```

هنگامی که یک تگ توسط گوشی شناسایی می‌شود، یک Intent به برنامه فرستاده می‌شود. این Intent و اطلاعات آن باید توسط برنامه handle شود. پس تابع handleIntent را به گونه‌ای می‌نویسیم که اطلاعات دریافت شده را پردازش کند. این تابع را در onCreate فراخوانی می‌کنیم:

```
handleIntent getIntent();
```

در تابع handleIntent دو شرط را باید بررسی کنیم: شرط اول این که Action موجود در Intent برابر با ACTION_NDEF_DISCOVERED باشد. شرط دوم این که اطلاعات موجود در این NDEF، از جنس plain/text یعنی متنی باشد. برای خواندن اطلاعات روی تگ دو روش داریم.

– روش اول: می‌توانیم به صورت مستقیم با تگ connect شویم و از متدهای زیر استفاده کنیم:

```
NdefMessage ndefMessage = ndef.getNdefMessage();
```

برای استفاده از این متدهای یک کلاس AsyncTask استفاده کنیم. علت این امر این است که هنگام پردازش اطلاعات، UI Thread درگیر نباشد و برنامه freeze نشود. برای حجم کم اطلاعات این کار لزومی ندارد، اما اگر حجم اطلاعات زیاد باشد، لازم است که پس از انجام انتقال اطلاعات و پردازش، UI Thread به روز رسانی شود.

– روش دوم: از اطلاعات cache شده استفاده کنیم. هنگام شناسایی یک تگ، اطلاعات آن دریافت شده و در cache ذخیره می‌شود. برای این کار از متدهای زیر استفاده می‌کنیم:

```
NdefMessage ndefMessage = ndef.getCachedNdefMessage();
```

برای فراخوانی این تابع، نیازی نیست که از UI Thread خارج شویم.

در این برنامه، به دو دلیل از روش دوم استفاده کرده ایم. اول آن که اطلاعات تگ ثابت است و

همان اطلاعات cache شده می‌تواند مورد استفاده قرار بگیرد. و دلیل دوم آن که حجم اطلاعات کم است و نیازی به آپدیت UI بعد از انتقال اطلاعات نیست.

پس تابع handleIntent به صورت زیر نوشته می‌شود:

```
NdefMessage ndefMessage = ndef.getCachedNdefMessage();
NdefRecord[] records = ndefMessage.getRecords();
for (NdefRecord ndefRecord : records) {
    if (ndefRecord.getTnf() == NdefRecord.TNF_WELL_KNOWN
        &&
        Arrays.equals(ndefRecord.getType(), NdefRecord.RTD_TEXT)) {
        result = new String(ndefRecord.getPayload());
    }
}
```

در این تابع، ابتدا NdefMessage از cache خوانده می‌شود. سپس Record های این پیغام بررسی شده و Payload آن Record ی که پیغام در آن ذخیره شده است، در برنامه نمایش داده می‌شود. برای آن که Record مورد نظر که حاوی پیغام Tag است را پیدا کنیم، لازم است نوع Record بررسی شود.

چون در Tag ها به دنبال اطلاعات متنی هستیم، باید نوع TNF در Record برابر با TNF WELL KNOWN و نوع RTD آن نیز برابر با RTD TEXT باشد. چگونگی بررسی این شرایط در تابع بالا قابل مشاهده است. (RTD به معنای Record Type Definition و TNF به معنای Format است).

نکته‌ی دیگر در مورد این اپلیکیشن، این است که اگر باز باشد و یک تگ شناسایی شود، به جای آن که اطلاعات تگ جدید در همین برنامه خوانده شود، یک نمونه از این اپلیکیشن مجددا باز می‌شود. یعنی به ازای هر بار خواندن تگ، یک برنامه‌ی جدید روی صفحه می‌آید. برای رفع این مشکل، باید از Foreground Dispatch استفاده کنیم. این کار باعث می‌شود که هرگاه intent جدیدی دریافت شود که برنامه قادر به handle کردن آن است، به جای باز کردن برنامه‌ی جدید، تابع handleIntent از برنامه‌ی جاری فراخوانی شود.

برای این کار، باید تنها هنگامی که برنامه باز است، Foreground Dispatch را فعال کنیم و در غیر

این صورت غیر فعال باشد. بنابراین در متدها onResume و onPause آن را enable کرده و در متدها onPause و onResume آن را disable می‌کنیم.

در تابع dispatchPendingIntent instance از Foreground Dispatch enable تعریف می‌کنیم و در صورتی که یک Intent دریافت شد، آن را با جزئیات آن Intent پر می‌کنیم:

```
final PendingIntent pendingIntent =
    PendingIntent.getActivity(activity.getApplicationContext(), 0, intent, 0);
```

همچنین برای تشخیص این که آیا برنامه قادر به handle کردن Intent ورودی هست یا خیر، یک IntentFilter تعریف می‌کنیم و مشخصات Intent را با آن مطابقت می‌دهیم:

```
IntentFilter[] filters = new IntentFilter[1];

filters[0] = new IntentFilter();
filters[0].addAction(NfcAdapter.ACTION_NDEF_DISCOVERED);
filters[0].addCategory(Intent.CATEGORY_DEFAULT);
filters[0].addDataType(MIME_TEXT_PLAIN);

String[][] techList = new String[][] {};
```

اگر برنامه نمی‌توانست Intent ورودی را handle کند، Intent Dispatch System به Intent داده می‌شود.

و در انتهای تابع setupForegroundDispatch Foreground Dispatch را فعال می‌کنیم:

```
adapter.enableForegroundDispatch(activity, pendingIntent, filters, techList);
```

به این صورت برنامه کامل می‌شود.

۳-۵ مژول خواندن تگ QRCode

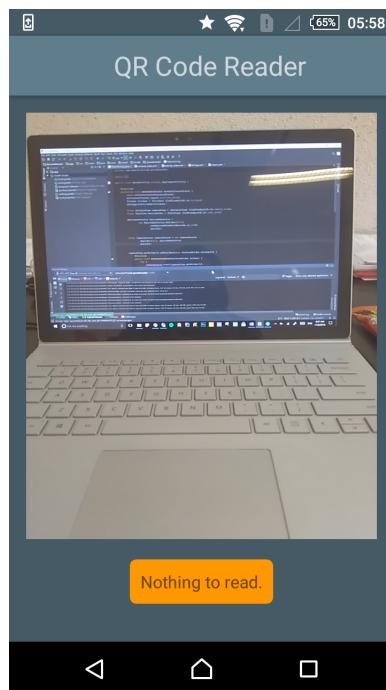
۱-۳-۵ معرفی مژول

این مژول برای گوشی‌های اندرویدی با ورژن اندروید ۴.۰ (API 14) و بالاتر نوشته شده است. هدف این اپلیکیشن، خواندن اطلاعات QR Code ها و نمایش این اطلاعات روی صفحه است. این اپلیکیشن برای خواندن QR Code ها از دوربین گوشی استفاده می‌کند و با استفاده از Google Play Services نوشته شده است.

۲-۳-۵ نمونه‌ای از کارکرد برنامه

هنگامی که کاربر برنامه را باز می‌کند با صفحه‌ای مانند شکل ۱۶-۵ مواجه می‌شود. در این صفحه کادری برای نمایش تصاویر ثبت شده توسط گوشی، و یک کادر نارنجی رنگ برای نمایش محتوای اطلاعات QR Code وجود دارد.

پس از تشخیص یک QR Code با کمک دوربین، اطلاعات آن QR Code به صورت خودکار در کادر نارنجی رنگ نمایش داده می‌شود و تا زمانی که یک QR Code با اطلاعات دیگر تشخیص داده نشود، این اطلاعات روی صفحه باقی می‌ماند. (شکل ۱۷-۵)



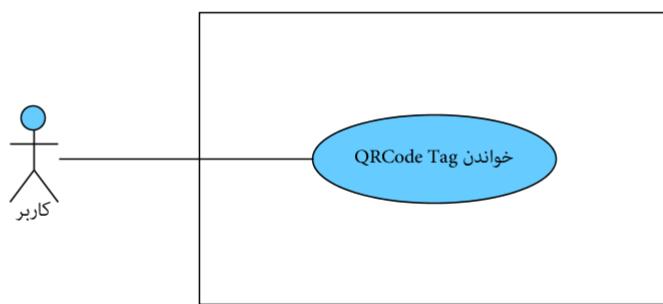
شکل ۱۶-۵: تصویری از صفحه‌ی اصلی برنامه.



شکل ۱۷-۵: تصویری از نمایش اطلاعات یک تگ در صفحه‌ی اصلی برنامه.

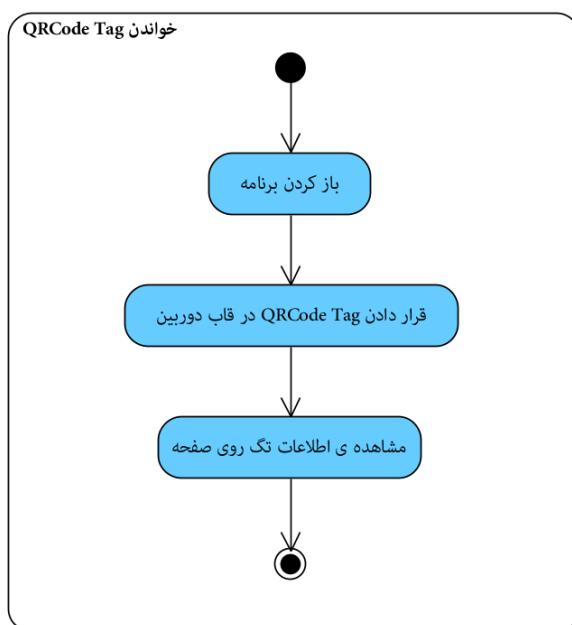
۳-۳-۵ نمودارهای تحلیل و طراحی

این برنامه شامل یک مورد کاربرد ساده یعنی خواندن تگ QRCode است و نمودار آن در تصویر ۱۸-۵ قابل مشاهده است.



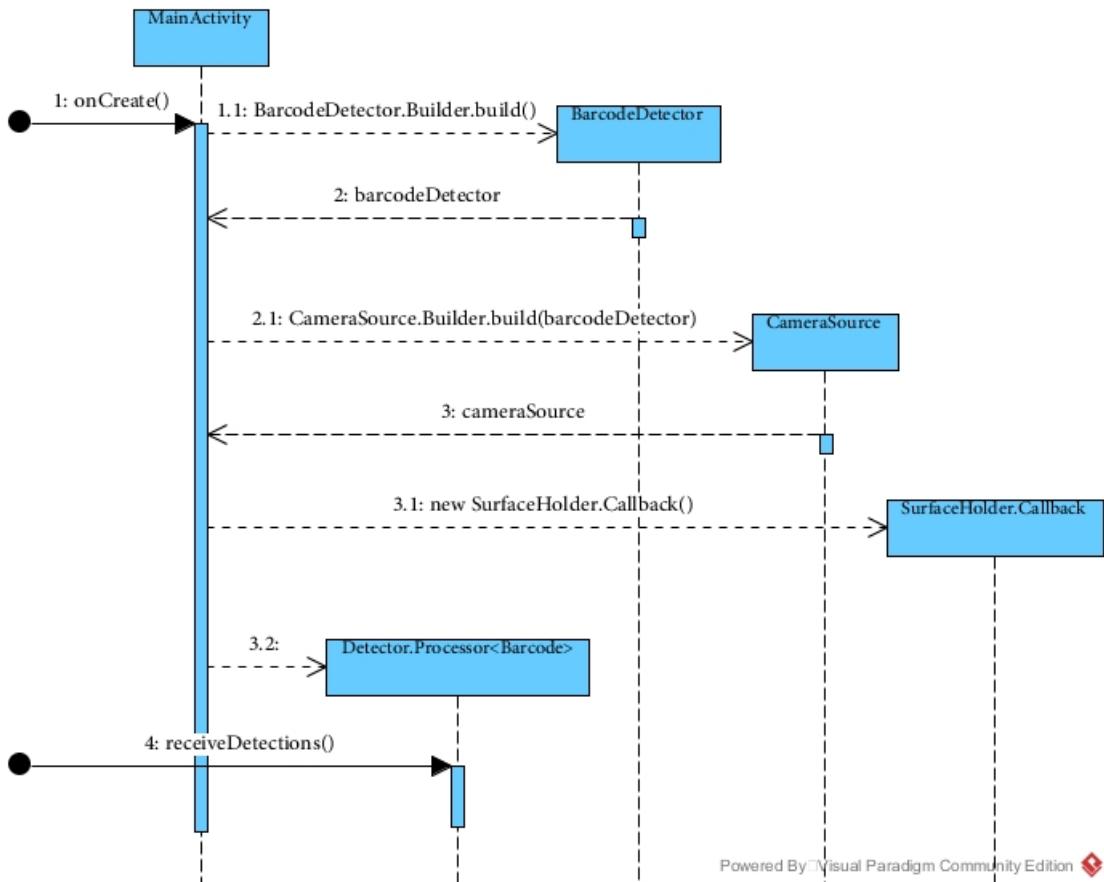
شکل ۱۸-۵ : نمودار کاربرد برنامه‌ی QRCode.

نمودار فعالیت تنها مورد کاربرد این برنامه نیز در تصویر ۱۹-۵ آمده است.



شکل ۱۹-۵ : نمودار کاربرد برنامه‌ی QRCode.

نمودار توالی کلاس اصلی برنامه در شکل ۲۰-۵ آمده است.



شکل ۲۰-۵: تصویری از نمودار توالی برنامه‌ی QRCode.

۴-۳-۵ پیاده‌سازی

برای تشخیص انواع Barcode در اندروید، کلاس BarcodeDetector در Vision API تعریف شده است. با استفاده از Builder این کلاس، می‌توان از این کلاس Instance گرفت. به صورت پیشفرض، این کلاس تمامی انواع barcode را تشخیص می‌دهد. اما برای داشتن performance بهتر، توصیه شده است که اگر هدف برنامه تشخیص نوع خاصی از Barcode هاست، آن را هنگام ساخت instance ذکر کنیم.

برای تشخیص QR، نمونه‌ی کلاس به شکل زیر در متد onCreate() ساخته می‌شود:

```
BarcodeDetector barcodeDetector =
    new BarcodeDetector.Builder(this)
        .setBarcodeFormats(Barcode.QR_CODE)
        .build();
```

برای هماهنگی Detector با دوربین گوشی، باید از کلاس CameraSource استفاده کنیم. این کلاس در فواصل زمانی مشخص تصویر ثبت شده توسط دوربین را به Detector ارسال می‌کند. در ادامه‌ی متد onCreate() یک نمونه از کلاس CameraSource می‌سازیم و در بالا ساختیم را به عنوان Detector برای آن مشخص می‌کنیم:

```
final CameraSource cameraSource = new CameraSource
    .Builder(this, barcodeDetector)
    .build();
```

برای نمایش تصاویر دوربین به صورت Real Time به یک SurfaceView در UI احتیاج داریم. با کمک کد زیر به این element دسترسی پیدا می‌کنیم:

```
final SurfaceView cameraView = (SurfaceView) findViewById(R.id.camera_view);
```

حال باید ارتباط بین این SurfaceView و CameraSource را برقرار کنیم. برای این کار، به یک CameraSource، SurfaceView را start کنند: callback

```

cameraView.getHolder().addCallback(new SurfaceHolder.Callback() {

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        try {
            cameraSource.start(cameraView.getHolder());
        } catch (IOException ie) {
            Log.e("CAMERA SOURCE", ie.getMessage());
        }
    }
})

```

و قدم آخر این است که هنگام تشخیص یک barcode توسط BarcodeDetector، اطلاعات خوانده شده را در صفحه نمایش بدهیم. برای این کار، از تابع setProcessor در BarcodeDetector استفاده می‌کنیم. این تابع یک کلاس Processor از جنس Detector را به عنوان ورودی دریافت می‌کند که تابعی را هنگام تشخیص یک barcode فراخوانی می‌کند. کافیست در این تابع، اطلاعات خوانده شده را در TextView نمایش بدهیم:

```

barcodeDetector.setProcessor(new Detector.Processor<Barcode>() {
    @Override
    public void receiveDetections(Detector.Detections<Barcode> detections) {
        final SparseArray<Barcode> barcodes = detections.getDetectedItems();
        if (barcodes.size() != 0) {
            barcodeInfo.post(new Runnable() {
                public void run() {
                    barcodeInfo.setText(barcodes.valueAt(0).displayValue);
                }
            });
        }
    }
});

```

نکته‌ی قابل توجه در مورد کد بالا این است که امکان کار با `receiveDetections` در تابع `UI Thread` ندارد. پس برای تغییر در محتوای `TextView`، از یک کلاس `Runnable` استفاده کرده‌ایم.

۴-۵ ماثول گام‌شمار

۱-۴-۵ معرفی ماثول

ماثول گام‌شمار برای ساعت‌های هوشمند اندرویدی با ورژن Android Wear 1.0 طراحی شده است. این ماثول می‌تواند تعداد تقریبی گام‌های کاربر را اعلام کند. کار این ماثول فعال کردن حسگر Step Counter و خواندن اطلاعات آن است. این حسگر به صورت Built In در برخی از ساعت‌های هوشمند وجود دارد.

۲-۴-۵ نمونه‌ای از کارکرد برنامه

در تنها صفحه‌ی این برنامه یک شمارش‌گر وجود دارد که تعداد گام‌ها را نشان می‌دهد. این عدد در ابتدا صفر است و با خاموش شدن ساعت نیز برابر با صفر قرار می‌گیرد. در غیر این صورت این عدد مقدار قبلی را ذخیره کرده و با هر گام آن را افزایش می‌دهد.

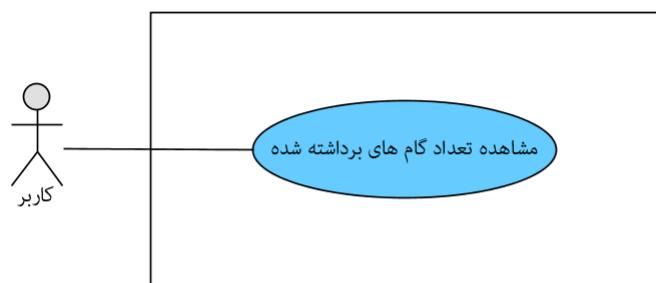
در شکل ۲۱-۵ تصویری از این برنامه نشان داده شده است.



شکل ۲۱-۵: تصویری از صفحه‌ی برنامه‌ی گام‌شمار.

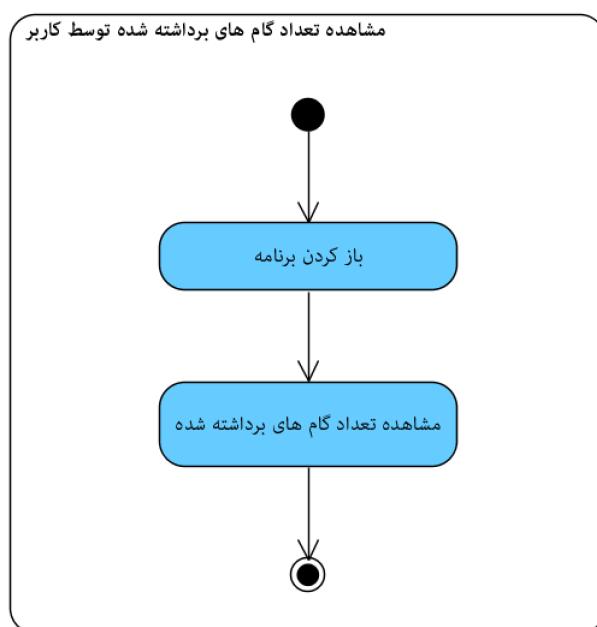
۳-۴-۵ نمودارهای تحلیل و طراحی

نمودار مورد کاربرد برای این برنامه بسیار ساده است و در شکل ۲۲-۵ نشان داده شده است.



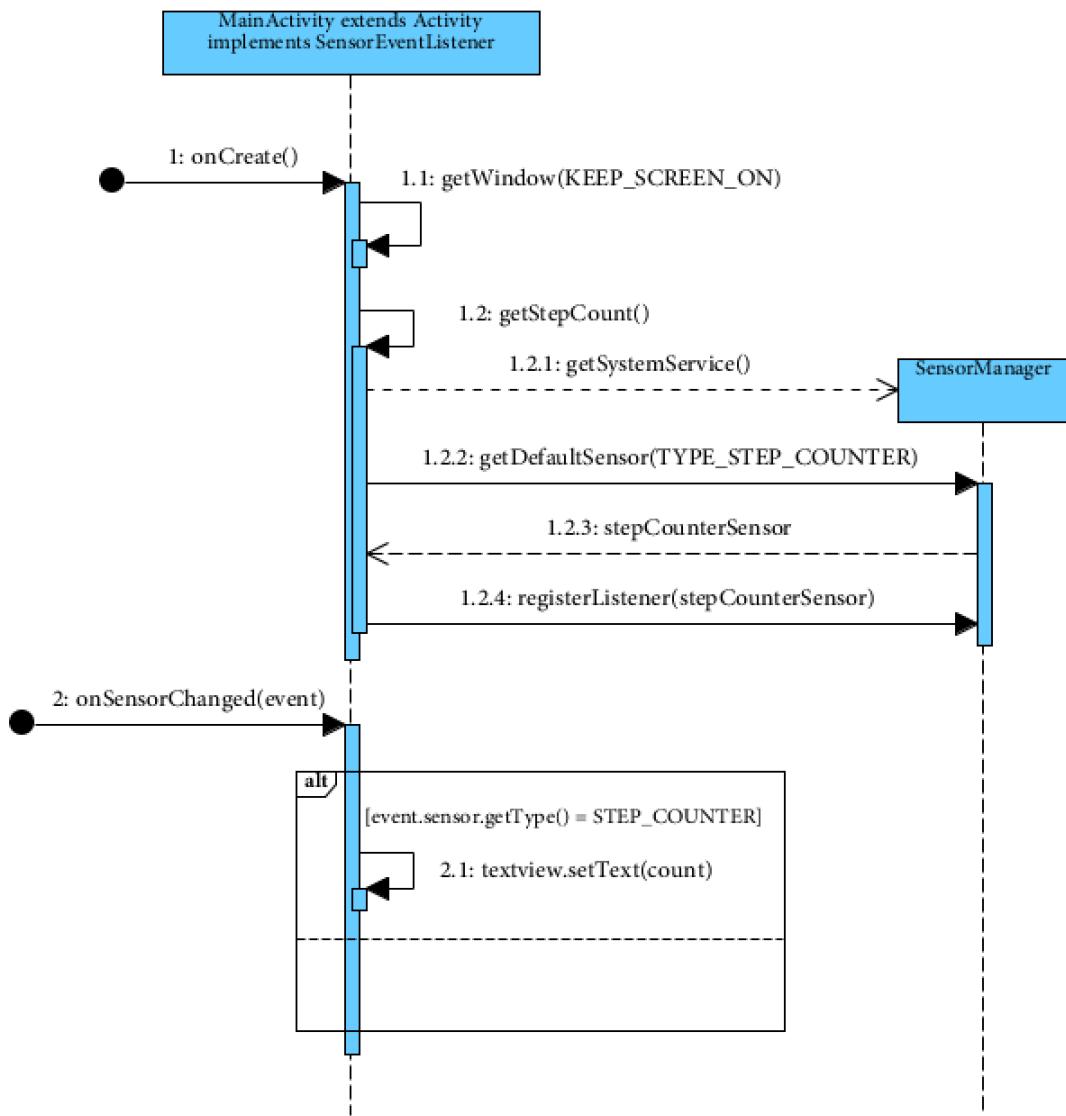
شکل ۵-۲۲: نمودار مورد کاربرد برنامه گام‌شمار.

نمودار فعالیت این برنامه شامل دو مرحله‌ی باز کردن برنامه و مشاهده تعداد گام‌هاست. این نمودار در تصویر ۲۳-۵ آمده است.



شکل ۵-۲۳: نمودار فعالیت برنامه گام‌شمار.

در نمودار توالی این برنامه (۲۴-۵) دو کلاس اصلی `MainActivity` و `SensorManager` و پیغام‌های ارسالی بین این دو کلاس دیده می‌شود. در قسمت پیاده‌سازی به شرح جزیات این پیغام‌ها می‌پردازیم.



شکل ۵-۲۴: نمودار توالی برنامه‌ی گام‌شمار.

۴-۴-۵ پیاده‌سازی

به طور کلی در برنامه‌های اندرویدی برای ارتباط با یک حسگر باید یک Listener تنظیم کنیم که تغییرات مقادیر خوانده شده توسط حسگر را رصد کند. برای این کار، کلاس MainActivity که کلاس اصلی این برنامه است، کلاس SensorEventListener را implement می‌کند:

```
public class MainActivity extends Activity implements SensorEventListener
{...}
```

در متدهای onCreate() که با شروع برنامه اجرا می‌شود، ابتدا دستور زیر را برای روش نگه داشتن صفحه اضافه می‌کنیم. برنامه‌های ساعت‌های هوشمند به طور پیش‌فرض نور صفحه را پس از مدت کوتاهی خاموش می‌کنند تا در مصرف باتری صرفه‌جویی شود. برای این که صفحه روشن بماند و تعداد گام‌ها را به صورت لحظه‌ای مشاهده کنیم، لازم است که پرچم FLAG KEEP FLAG که از getStepCount() امکان SCREEN ON را فعال کنیم. همچنین در این تابع، با صدا زدن تابع() امکان برقراری ارتباط با حسگر را ایجاد می‌کنیم:

```
getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
getStepCount();
```

در تابع() ابتدا دسترسی به حسگر Step Counter را از سیستم می‌گیریم. سپس کلاس MainActivity را به عنوان Listener برای این حسگر تنظیم می‌کنیم:

```
private void getStepCount() {
    SensorManager mSensorManager = ((SensorManager)
        getSystemService(SENSOR_SERVICE));
    Sensor mStepCountSensor =
        mSensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
    mSensorManager.registerListener(this, mStepCountSensor,
        SensorManager.SENSOR_DELAY_NORMAL);
}
```

با تنظیم این کلاس به عنوان Listener، می‌توانیم محتوای دو تابع() و onAccuracyChanged() را

که مربوط به حسگر هستند را بازنویسی کنیم. در اینجا چون وارد بحث onSensorChanged() دقت حسگر نشده‌ایم، از تابع onAccuracyChanged() استفاده نمی‌کنیم. اما تابع onSensorChanged() کار اصلی را برای ما انجام می‌دهد. این تابع به هنگام بروز تغییر در حسگرها توسط سیستم فراخوانی می‌شود. پس برای مشاهده‌ی اطلاعات جدید در حسگر، کافی است که هنگام فراخوانی این تابع تعداد گام‌ها را به‌روزرسانی کنیم.

```
public void onSensorChanged(SensorEvent event) {  
    if (event.sensor.getType() == Sensor.TYPE_STEP_COUNTER) {  
        String msg = String.valueOf((int) event.values[0]);  
        mTextViewStepCount.setText(msg);  
    }  
}
```

به این صورت با هر بار تغییر مقدار حسگر، تعداد گام‌ها روی صفحه به‌روزرسانی شده و برنامه تکمیل می‌شود.

۵-۵ مازول شمارندهی حرکت پروانه

۱-۵-۵ معرفی مازول

مازول شمارندهی حرکت پروانه برای ساعت‌های هوشمند اندرویدی با ورژن ۱.۰ Android Wear طراحی شده است. هدف این اپلیکیشن، تشخیص یک نوع حرکت ورزشی است که اصطلاحاً حرکت پروانه نامیده می‌شود. کاربر با بستن ساعت مچی هوشمند و راه اندازی این اپلیکیشن می‌تواند تعداد انجام این حرکت را بشمارد. اساس کار این مازول مبتنی بر بررسی تغییرات مقادیر حسگر جاذبه یا Gravity Sensor است.

۲-۵-۵ نمونه‌ای از کارکرد برنامه

در تنها صفحه‌ی این برنامه یک شمارش‌گر وجود دارد که تعداد حرکت‌های پروانه را نشان می‌دهد. این عدد در ابتدا صفر است و با هر حرکت پروانه‌ای که تشخیص داده می‌شود یک واحد افزایش می‌یابد.

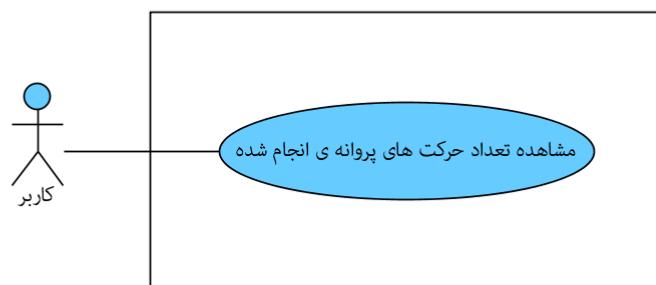
در شکل ۲۵-۵ تصویری از این برنامه نشان داده شده است.



شکل ۲۵-۵: تصویری از صفحه‌ی برنامه‌ی شمارندهی حرکت پروانه.

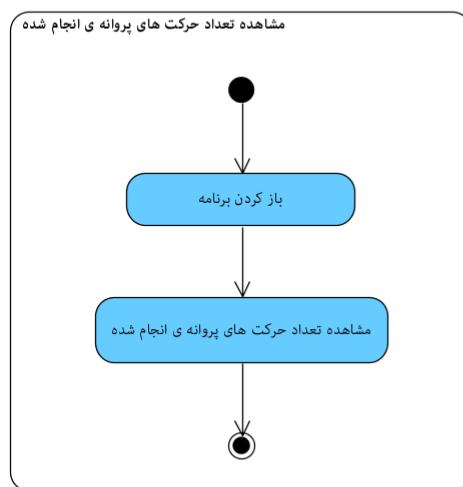
۳-۵-۵ نمودارهای تحلیل و طراحی

نمودار مورد کاربرد برای این برنامه بسیار ساده است و در شکل ۲۶-۵ نشان داده شده است.



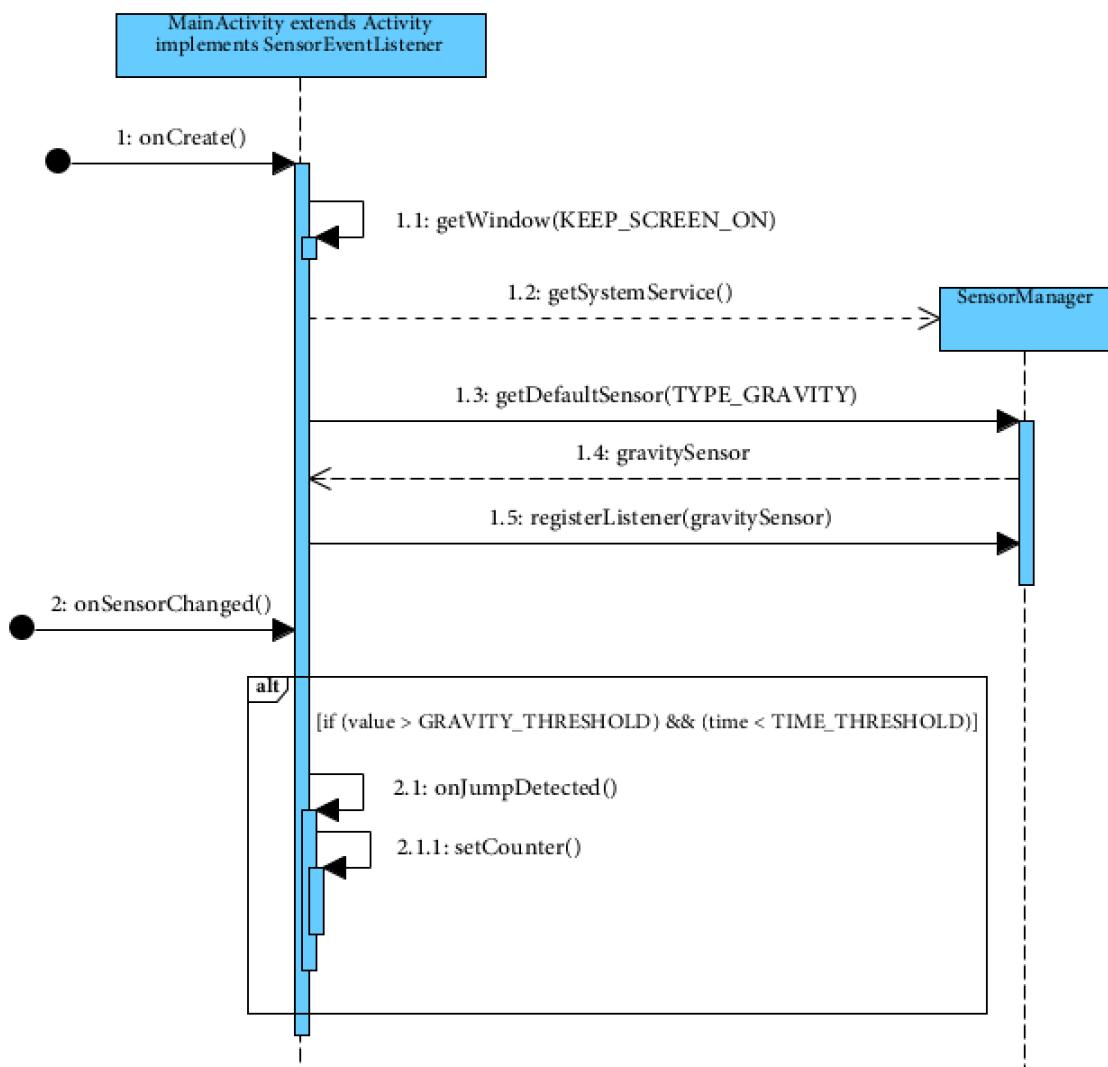
شکل ۲۶-۵: نمودار مورد کاربرد برنامه‌ی شمارندهی حرکت پروانه.

نمودار فعالیت این برنامه شامل دو مرحله‌ی باز کردن برنامه و مشاهده‌ی تعداد حرکت‌های پروانه است. این نمودار در تصویر ۲۷-۵ آمده است.



شکل ۲۷-۵: نمودار فعالیت برنامه‌ی شمارندهی حرکت پروانه.

در نمودار توالی، پیغام‌های ارسالی بین دو کلاس اصلی برنامه دیده می‌شود. در قسمت پیاده‌سازی به شرح جزییات این پیغام‌ها می‌پردازیم.



شکل ۵-۲۸: نمودار توالی برنامه‌ی شمارنده‌ی حرکت پروانه.

۴-۵-۵ پیاده‌سازی

این مازول بر اساس مقادیر خوانده شده از حسگر جاذبه، حرکت پروانه را تشخیص می‌دهد. می‌دانیم که شتاب جاذبه‌ی زمین مقداری برابر با ۹.۸ متر بر میزانه دارد. با تغییر جهت دست و در نتیجه تغییر جهت ساعت، مقدار شتاب اندازه‌گیری شده توسط حسگر نیز تغییر می‌کند. هنگامی که در یک حرکت دست به طرف بالا گرفته شود، مقدار شتاب جاذبه برابر است با منفی مقدار شتاب در حالتی که دست به طرف پایین گرفته شده است.

حال به دلیل این که ممکن است حسگر با مقداری خطأ همراه باشد و همچنین کاربر دستش را به صورت کامل به بالا و پایین حرکت ندهد، به جای ۹.۸ از یک مقدار کوچکتر استفاده می‌کنیم. برای مثال در اینجا از مقدار ۷ استفاده کردیم.

برای خواندن مقادیر از حسگر جاذبه، کافی است که دسترسی به این حسگر را از کلاس SensorManager بگیریم و برای آن یک Listener تنظیم کنیم. برای این کار، کلاس MainActivity که کلاس اصلی این برنامه است، کلاس SensorEventListener را implement می‌کند:

```
public class MainActivity extends Activity implements SensorEventListener
{...}
```

با شروع برنامه، متده است onCreated() اجرا می‌شود. در ساعت‌های هوشمند به دلیل صرفه‌جویی در مصرف باتری مدت زمان روشن ماندن صفحه کوتاه است. اگر بخواهیم در برنامه‌ای این مدت زمان را افزایش دهیم باید به صورت دستی آن را تنظیم کنیم. دستور زیر این کار را انجام می‌دهد:

```
getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

برای دسترسی به داده‌های حسگر، با دستور زیر دسترسی به حسگر را از کلاس SensorManager گرفته و Listener را برای آن تنظیم می‌کنیم:

```
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY);
mSensorManager.registerListener(this, mSensor,
    SensorManager.SENSOR_DELAY_NORMAL);
```

با تنظیم Listener، در موقع تغییر مقدار حسگر تابع onSensorChanged() به صورت خودکار توسط سیستم فراخوانی می‌شود. پس کافی است که با بررسی این مقدار جدید بینیم که آیا حرکت پروانه رخ داده است یا خیر. طبق تعریف ما، حرکت پروانه زمانی رخ می‌دهد که مقدار حسگر جاذبه در کمتر از یک مدت زمان خاص به مقدار مشخصی تغییر کند. در کد زیر مدت زمان را با TIME_THRESHOLD و مقدار تغییر را با GRAVITY_THRESHOLD نشان داده‌ایم. همچنین برای این که هر حرکت کامل شامل دو بار حرکت دست است، متغیر xValue را تعریف کرده‌ایم که هر دو بار حرکت دست به یک بار شمارش حرکت منجر شود:

```

@Override
public void onSensorChanged(SensorEvent event) {
    long timestamp = event.timestamp;
    if ((Math.abs(xValue) > GRAVITY_THRESHOLD)) {
        if (timestamp - mLastTime < TIME_THRESHOLD_NS && mUp != (xValue > 0)) {
            onJumpDetected(!mUp);
        }
        mUp = xValue > 0;
        mLastTime = timestamp;
    }
}

```

در کد بالا دیدیم که با هر بار تشخیص حرکت پروانه، تابع onJumpDetected() صدای زده می‌شود. در این تابع اگر جهت حرکت به سمت بالا باشد، هیچ کاری صورت نمی‌گیرد و از تابع خارج می‌شود. اما اگر جهت حرکت به سمت پایین باشد شمارش گر یک واحد افزایش می‌یابد و نمایش آن روی صفحه نیز به روز رسانی می‌شود.

```

void onJumpDetected(boolean up) {
    if (up) {
        return;
    }
    mJumpCounter++;
}

```

```
setCounter(mJumpCounter);  
renewTimer();  
}
```

به این صورت تعداد حرکت‌های پروانه‌ی انجام شده توسط کاربر روی صفحه نشان داده می‌شود.

فصل ۶

نتیجه‌گیری

در این تحقیق سعی کردیم حوزه‌ی نسبتاً جدید بازی‌های واقعیت فراگیر را معرفی کنیم و به بحث در مورد جنبه‌های مختلف آن پردازیم. سناریوی نمونه‌ای برای انتقال بهتر مفهوم شرح دادیم و به تحلیل مختصراً از فعالیت‌هایی که تا کنون در این حوزه انجام شده پرداختیم. یک معماری پیشنهادی برای پیاده‌سازی بازی‌های واقعیت فراگیر معرفی کردیم. با تمرکز بر قسمت تعامل کاربر با بازی، راه‌های موجود را امکان‌سنجی کردیم. و در نهایت با طراحی و پیاده‌سازی پنج مژول سعی کردیم ابزاری برای ساخت هرچه بهتر و سریع‌تر این نوع بازی ارائه دهیم.

کتاب نامه

- [1] S. Hinske, M. Lampe, C. Magerkurth, and C. Rocker. Classifying pervasive games: On pervasive computing and mixed reality. *Aachen: Shaker*, 1(2):10–18, 2007.
- [2] C. Ardito, R. Lanzilotti, D. Raptis, C. Sintoris, N. Yiannoutsou, and M. Avouris. Designing pervasive games for learning. *Design, User Experience, and Usability. Theory, Methods, Tools and Practice - First International Conference, DUXU 2011*, 2011.
- [3] D. Spikol, O. Pettersson, and A. Gerestrand. Designing pervasive games to support university studies in media technology. *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, 2009.
- [4] C. Sintoris, N. Yiannoutsou, S. Demetriou, and N. Avouris. Discovering the invisible city: Location-based games for learning in smart cities. 2013.
- [5] M. Bang, A. Gustafsson, and C. Katzeff. Promoting new patterns in household energy consumption with pervasive learning games. *Lecture Notes in Computer Science, vol 4744. Springer, Berlin, Heidelberg*, 2007.
- [6] D. Linner, F. Kirsch, I. Radusch, and S. Steglich. Context-aware multimedia provisioning for pervasive games. *Seventh IEEE International Symposium on Multimedia, Irvine, CA, USA*, 2006.
- [7] M. Roj. Smart artifacts as a key component of pervasive games. *Mobile and Embedded Applications Group, Institute of Telecommunications, Warsaw University of Technology*, 2004.
- [8] S. Lundgren and S. Björk. Game mechanics: Describing computer-augmented games in terms of interaction. 2003.