Міністерство освіти і науки України
Сумський державний педагогічний університет імені А.С. Макаренка
Фізико-математичний факультет
Кафедра інформатики

Мигаль Віталій Олександрович

ОСОБЛИВОСТІ НАВЧАННЯ МАЙБУТНІХ УЧИТЕЛІВ ІНФОРМАТИКИ ВІЗУАЛЬНОМУ ПРОГРАМУВАННЮ(НА ПРИКЛАДІ СЕРЕДОВИЩА MIT APP INVENTOR)

Спеціальність: 014.08 Середня освіта (інформатика)
Галузь знань: 01. Освіта

Кваліфікаційна робота
на здобуття освітнього ступеня магістра

Науковий керівник:
_____ Н.В. Дегтярьова
доцент
«___»_____ 2020 року
Виконавець
_____ В.О. Мигаль
«___»_____ 2020 року

Суми 2020

ЗМІСТ

ВСТУП

Актуальність дослідження: У сучасному інформаційному світі вміння програмувати є доволі корисним. Знайти роботу з високою заробітною платою досить легко, якщо є гарні навички в програмуванні. Проте більшості людей здається складним навчання програмуванню. Для таких візуальне програмування може показати, що не все так складно, адже освоїти візуальну мову програмування легше ніж текстову. Тому актуальність даної теми є досить високою.

Візуальне програмування було створено для того, щоб спростити створення та розробку програм. А тому використання таких мов - спрощує навчанню програмуванню. Використання таких середовищ навчає не тільки візуальному програмуванню, а і програмуванню в цілому.

Візуальне програмування студенти опановували ще у школі вивчаючи Scratch. Середовище MIT App Inventor 2 логічно продовжує дану лінію, тому його простіше освоїти. Працюючи в App Inventor 2 студенти створюють додатки на ОС Android, що можна використати як додаткову мотивацію до вивчення цього середовища.

Оскільки вивчення курсу полягає в набутті практичних навичок візуального програмування, то теоретичний матеріал, який подається, треба теж практично закріплювати. Після вивчення нового матеріалу студенти повинні самостійно з ним попрацювати, ознайомитися, адже в них можуть виникнути питання.

На лабораторних заняттях потрібно чітко пояснювати та показувати завдання, яке необхідно зробити , спостерігати за процесом роботи виконання завдань. Викладач повинен бути поряд та готовим відповісти на питання. Також спостерігаючи за студентами можна побачити як добре кожен з них оволодів середовищем та консультувати якщо це необхідно. Це покращує закріплення матеріалу студентом. Потрібно пам’ятати, що вивчення цього курсу вплине на те, чи цікавим буде студентам програмування в подальшому.

Об’єкт дослідження: візуальне програмування.

Предмет дослідження: навчання майбутніх учителів інформатики візуальному програмуванню.

Метою роботи є дослідження реалізації навчання майбутніх учителів інформатики візуальному програмуванню за допомогою середовища App Inventor 2.

Для реалізації поставленої мети розв'язувались наступні **завдання:**

1) вивчення матеріалів щодо візуального програмування;

2) ознайомлення та вивчення середовища візуального програмування MIT App Inventor 2;

3) проаналізувати науковість та методичні матеріали стосовно викладання візуального програмування у ЗВО;

4) розробити лекційні заняття та лабораторні роботи для вивчення поняття візуальне програмування та роботу в MIT App Inventor 2.

Методи дослідження: для проведення дослідження використано теоретичні (аналіз науково-методичної літератури, аналіз методів і принципів навчання, аналіз методів навчання інформатики, порівняльний аналіз для з'ясування різних поглядів на проблему) та емпіричні (спостереження, бесіди) методи навчання.

Апробація результатів дослідження: результати дипломної роботи доповідалися на звітній науковій конференції студентів фізико-математичного факультету (СумДПУ ім. А.С. Макаренка, листопад 2020) а також на міжнародній науково-практичній конференції "Modern problems in science" (Прага, Чехія, 09-12 листопада 2020).

Практична значущість дослідження полягає в розробці лекцій та лабораторних робіт для студентів першого курсу ЗВО.

Структура роботи: робота складається зі вступу, двох розділів, висновків та списку використаних джерел.

РОЗДІЛ І
ВІЗУАЛЬНЕ ПРОГРАМУВАННЯ

1.1 Візуальне програмування

Програмування - це процес, який дозволяє створювати програмне забезпечення та веб-сайти. В даний час комп'ютери не здатні думати самостійно; вони вимагають, щоб користувачі давали їм набори упорядкованих інструкцій, для виконання. Це називається «кодом». Більшість ресурсів, які використовуються на комп'ютері та в Інтернеті, зроблені за допомогою коду. Програмування є ключовим елементом навчальної програми з цифрових технологій, оскільки воно допомагає студентам розвинути такі важливі навички, як вирішення проблем, логіка і критичне мислення.[1]

Коли програмісти пояснюють комусь, що повинна робити програма, вони часто використовують дошку і графічне представлення потоку управління з прямокутниками і стрілками. Якщо таке уявлення так зручно, чому ми не можемо писати програми з їх допомогою? Ідея дуже приваблива, і в деяких областях візуальні мови програмування вже зробили собі ім'я. Зокрема, однією з перших областей, де вони досягли успіху, є викладання, оскільки вони можуть допомогти користувачам навчитися створювати комп'ютерні програми. Мета багатьох візуальних мов програмування полягає в тому, щоб зробити програмування більш доступним, зокрема, щоб зменшити труднощі, з якими стикаються новачки, коли вони починають програмувати.

Візуальне програмування - це вид програмування, що передбачає створення програм за допомогою наглядних засобів, тобто шляхом оперування графічними об'єктами, а не написання програмного коду в текстовому вигляді[2]. Візуальне програмування часто представляють як наступний етап розвитку текстових мов програмування. Основною суттю візуального програмування є побудова розв'язку поставленої задачі за допомогою візуальних заготовок, які вставляються у форму, присвоєння значень їхнім атрибутам і створення чи застосування потрібних для розв'язання даної задачі

методів. Останнім часом візуальному програмуванню стали приділяти більше уваги - у зв’язку з розвитком мобільних сенсорних пристроїв (смартфони, планшети). Візуальне програмування в основному використовується для створення програм з графічним інтерфейсом для операційних систем з графічним інтерфейсом користувача.[3]

Історія створення візуального програмування. Процес візуалізації використовувався майже століття і був розроблений ще до того, як був створений перший комп’ютер. У 1920-х роках нова форма планування набула першочергового значення, оскільки світ розвивався з великою швидкістю. Це призвело до підвищення інтересу в документуванні процесів промислового будівництва. Цілком природно, що ця документація була візуально представлена тим, що на сьогодні є блок-схема. Популярність цього методу зростала, і він став звичайним явищем при вирішенні складних завдань і підвищенні автоматизації. [4]

Тільки в 1949 році ця форма візуального програмування була реалізована в поєднанні з комп’ютерними програмами. Джон фон Нейман і Герман Гольдстайн хотіли правильно встановити кілька перемикачів кільцевого лічильника для управління входом і виходом. Маючи тисячі перемикачів для налаштування, вони прийняли систему блок-схем, щоб допомогти їм. Нейман і Голдстайн досягли успіху і довели, що візуальні мови програмування можна використовувати в швидко розвиваючому світі комп’ютерів.

Комп’ютерні вчені продовжували тестувати діапазон використання візуальних мов програмування, оскільки графічні можливості комп’ютерів збільшувалися. Один з таких тестів був проведений в 1963 році Іваном Сазерлендом. Для своєї дисертації він створив «Sketchpad», який був першим повним графічним інтерфейсом користувача. Він дозволяв користувачеві малювати на екрані і управляти малюнком на основі інформації, що вводиться через перемикачі. Ці креслення могли взаємодіяти один з одним і відповідати на кілька команд користувача. Деякі з них включають в себе зміну пропорцій

фігури, переміщення фігури, додавання і видалення частин фігури, а також перетягування частин фігури. Неймовірно, але навіть тоді комп’ютер виконував все у реальному часі.

Аналізуючи це можна зрозуміти, що основна мета раннього візуального програмування полягала в тому, щоб полегшити взаємодію з комп’ютером. Зрештою, перші графічні інтерфейси користувача (англ. graphical user interfaces) пропонували значно легший досвід користування у порівнянні з введенням рядків у вікно терміналу, до такої міри, що більшість людей сьогодні купуючи "Mac" або "PC" і навіть не уявляють свої комп’ютери відмінними від сучасних візуальних операційних систем.

Комп’ютерам тих часів все ще не вистачало потужного апаратного забезпечення, яке дозволяло б здійснювати значну взаємодію в реальному часі. Пройде не один рік, перш ніж у візуальному програмуванні з’явиться ще один серйозний прогрес. Тільки в 1975 році Девід Кенфілд Сміт опублікував свою власну дисертацію. Це була візуальна мова програмування, названа Пігмаліон (англ. Pygmalion, рис. 1.1).

Було заявлено, що Pygmalion - це мова, яка може взяти за основу природну творчість людського розуму і перевести її на абстрактну мову, зрозумілу комп’ютеру. Більше нікому не доведеться намагатися уявити собі, як комп’ютер розуміє програму. Користувачі зможуть побачити це у вигляді графічного знімка. Все, що потрібно було зробити користувачам - це змінити речі на основі наданого їм знімка. Pygmalion давав вихідний знімок, який описував його стан, а програмісти могли описати, як вони хотіли, щоб цей вихідний знімок змінився. Ця теза буде повторена в декількох майбутніх комп’ютерних мовах.

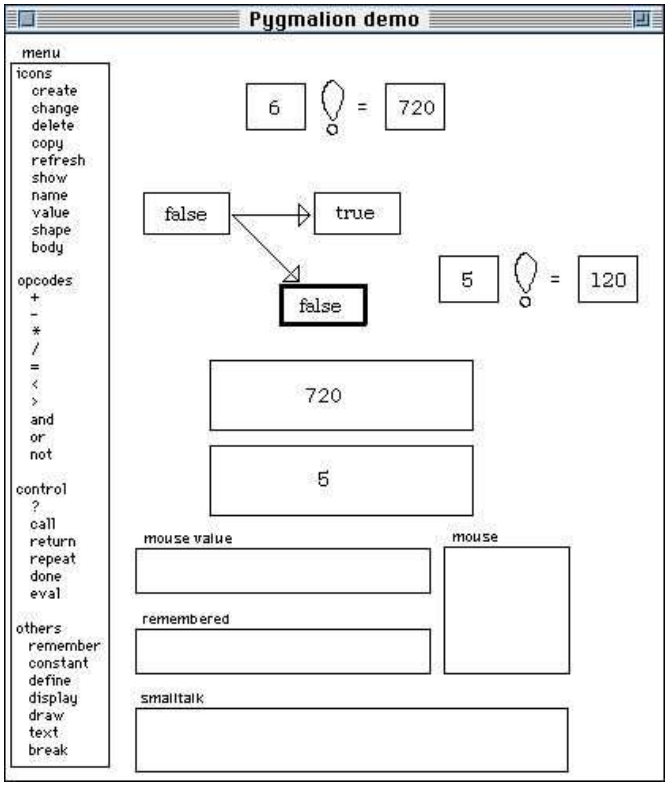

Рис. 1.1. Вигляд «Pygmalion».

Тільки в 1980-х роках комп'ютери стали більш популярними. Компанії по всьому світу керувалися за допомогою комп'ютерів. Їх також купували для домашнього використання. Нарешті, створювалися комп'ютери, які володіли достатньою потужністю, щоб обробляти візуальні програми, не займаючи занадто багато місця. Це призвело до того, що все більше і більше людей працювали над візуальними мовами програмування.[4]

Між 1982 і 1985 була задумана і розроблена «Prograph» для комп'ютерів Macintosh від Apple (рис. 1.2). Співробітники та студенти Університету Акадії погодилися, що діаграми набагато корисніші для позначення робочого процесу, і створили нову візуальну мову програмування, щоб довести це. Вони використовували об'єкти як форму спілкування. Ці об'єкти представляли собою шестикутники, розташовані на графіку. Користувачі могли взаємодіяти з ними, натиснувши на будь-яку зі сторін шестикутника. Одна сторона буде виробляти дані, а інша сторона буде надавати користувачеві інформацію про методи об'єкту. Графіки можуть бути сформовані між фігурами, щоб показати, як дані «перетікають» і як певні дії вплинуть на цей потік.[5]

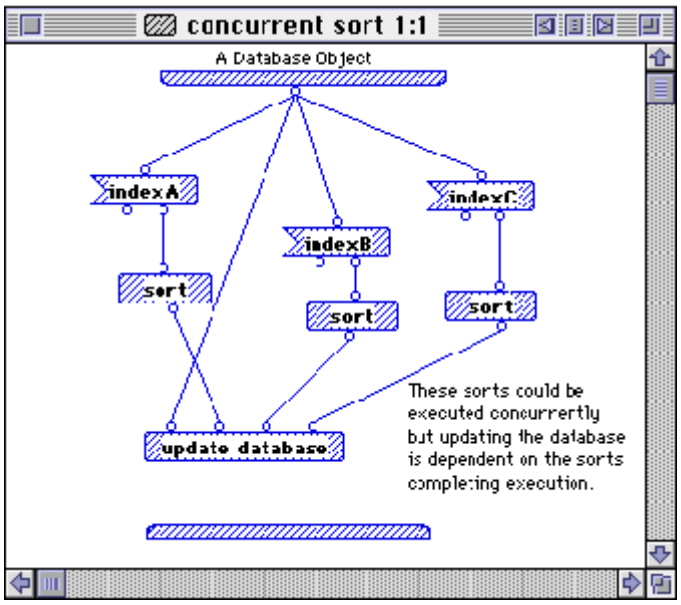

Рис. 1.2. «Prograph».

Здебільшого, візуальне програмування не було використано на комп'ютерах Windows в той час. У них просто не було графічної операційної системи, яка могла б обробляти сучасні візуальні мови програмування. З іншого боку, такі комп'ютери, як Commodore Amiga, Acorn Archimedes і Apple Mac, могли легко запускати ці програми.

На початку візуальні мови програмування отримали розвиток з того, що комп'ютерне обладнання ставало все більш потужним. Тим не менш, коли проблеми програмування стали складніше, у візуальних мов програмування також виникли труднощі. Були просто деякі речі, які було занадто важко зобразити візуально. Наприкінці 1990-х років цей вид програмування опинився в незрозумілому становищі. Він все ще був корисний, і він все ще був в змозі впоратися з деякими завданнями надзвичайно добре. Однак більшість сучасних комп'ютерів обрали шлях, за яким багато візуальних мов програмування не могли слідувати. Це призвело до затишшя у використанні візуального програмування, поки не були створені три нових гілки: мультимедіа, ігри та бізнес-системи.

Мультимедіа та візуальне програмування. За останні два десятиліття кількість мультимедійних продуктів збільшилася в геометричній прогресії. Це

може включати в себе що завгодно - від музики до ігор і енциклопедій на компакт-дисках. Такі компанії, як Phillips, Commodore, 3DO і Microsoft, прокладали шлях зі своїми мультимедійними плеєрами. Замість того, щоб просто використовувати текстовий код, вони використовували інтерактивні інструменти, які добре працювали з візуальними мовами програмування. Особливо це стосувалося музичних медіа.

Компанія Blue Ribbon SoundsWorks створила Bars and Pipes Professional, програма була MIDI-секвенсором (рис. 1.3). За допомогою неї користувачі створювали музику, взаємодіючи з прямокутними смугами, які представляли різні ноти на шкалі або різні інструменти. Синтезаторні програми, подібні до цієї, використовуються і донині.

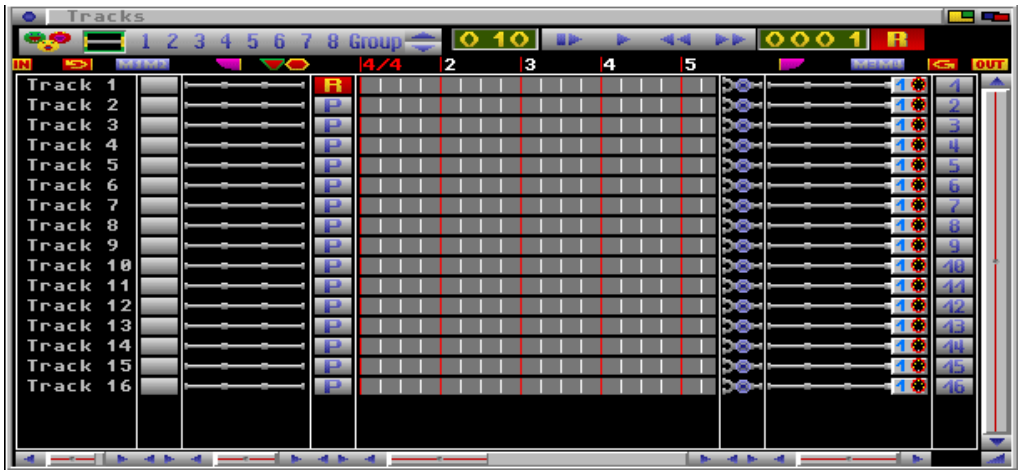

Рис. 1.3. «Bars&Pipes Professional»

Ігри та візуальне програмування

Коли користувач грає у відеоігри, йому може бути важко думати про це як про блок-схему. Однак вони саме такі. Майже кожну взаємодію в грі можна звести до твердження "якщо-то". «Якщо я натисну цю кнопку, то мій персонаж буде атакувати. Якщо я доторкнуся до ворога, то втрачу здоров’я. Якщо я буду рухатися в цьому напрямку, то навколишнє середовище зміниться». Ці твердження типу "якщо-то" є лише основою дуже складної блок-схеми. Те, якою могла бути гра, обмежувалося уявою її творця. З роками розробки програмного забезпечення відеоігри можна розділити на кілька різних категорій. Кожен з яких використовує свою власну форму візуальних мов

програмування. Таким чином, ці мови стали досить специфічними, наприклад Unreal Engine Blueprint (рис. 1.4) і Unity з Bolt.

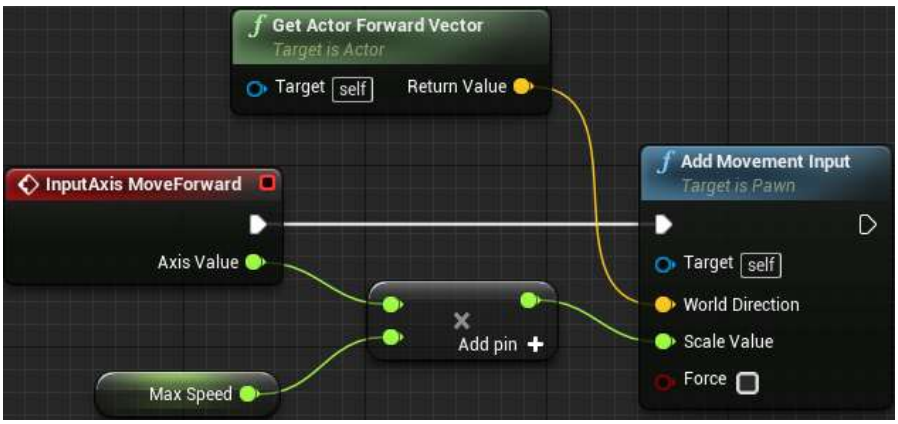

Рис. 1.4. «Unreal Engine Blueprints»

Бізнес-системи та візуальне програмування

Кожен бізнес має базу даних і процеси, які сприяють підвищенню ефективності і в кінцевому результаті прибутку. Бізнес-користувач може переміщатися по базі даних системи за допомогою процедурних мов або клієнтського програмного забезпечення. Цей процес зчитування даних або вказівки програмному забезпеченню переміщатися за потоком даних також схожий на блок-схему (рис. 1.5). Існує велика кількість інструментів і додатків, які допомагають бізнесу з моделюванням баз даних або процесів більш ефективно, ніж при використанні чистої мови програмування. Видатними категоріями в цьому секторі є ETL-інструменти і механізми документообігу.

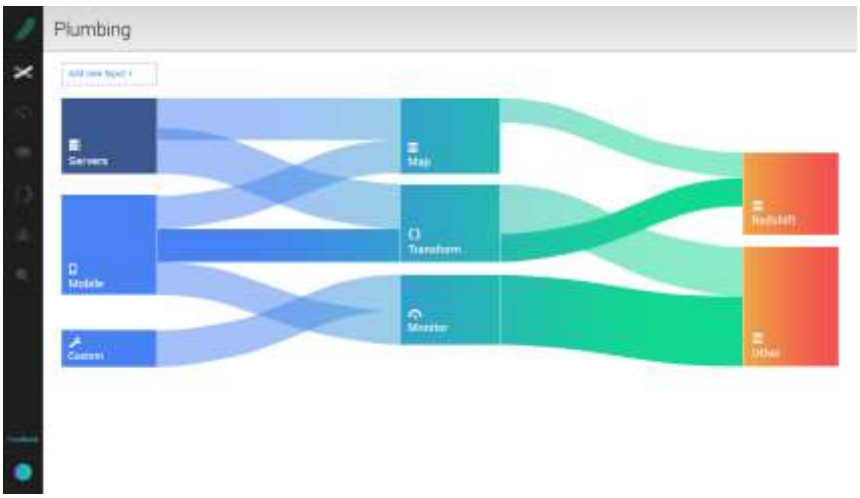

Рис. 1.5. Конвеєри даних у «Alooma»

Візуальне програмування виникло на основі об’єктно-орієнтованого програмування, як засіб автоматизації його процесів. Тепер для складання програми користувачу необхідно маніпулювати наданими графічними засобами - компонентами. Компоненти мають певні атрибути (властивості). Властивості можуть набувати значення зі заздалегідь фіксованого значення чи набору, або можуть бути придумані користувачем. Користувач розв’язує різноманітні задачі шляхом добирання компонентів надання потрібних значень їхнім атрибутам. При візуальному програмуванні програміст показує, що необхідно отримати в результаті, а текст програми генерується автоматично за допомогою візуального прототипу. Результати виводять на форму (вікно, характерне для операційної системи), де можна застосувати різноманітні елементи керування, властиві для діалогових вікон прикладних програм (кнопки, різноманітні поля).[6]

Працюючи у середовищах візуального програмування добре розвивається алгоритмічний тип або стиль мислення. У таких середовищах, при створенні програми, будується ланцюг дій, які буде виконувати програма у певному порядку, візуально це помітно. Тому при побудові програми, користувач розуміє, яку дію виконує той, чи інший елемент, та за чітко встановленим маршрутом.

Поняття "алгоритмічний стиль або тип мислення" широко використовується в сучасній методичній літературі, присвяченій навчанню інформатики. При цьому більшість авторів програм курсу шкільної інформатики вважає розвиток саме даного стилю мислення однією з основних цілей навчання інформатики. На жаль, автори в більшості випадків не визначають, що таке алгоритмічний стиль мислення. У кращому випадку цей термін пояснюється на емпіричному рівні. [7]

Даний стиль характеризується точністю, визначеністю, формальністю і, як правило, пов’язується з теоретичною діяльністю. Між тим алгоритмічний стиль мислення дозволяє розв’язувати задачі, що виникають у будь-якій сфері діяльності людини, а не лише в теоретичній, наприклад, програмуванні чи в

математиці, як традиційно вважається. Він не пов’язаний лише з обчислювальною технікою, бо саме поняття алгоритму, хоча й інтуїтивне, виникло задовго до появи першого комп’ютера. Розв’язуючи більшість задач, людина, в тій чи іншій мірі, застосовує алгоритмічний підхід, хоча окремі етапи цього процесу можуть носити асоціативний характер. Крім того, "алгоритмічний тип діяльності важливий не лише як потужний тип діяльності людини, а як одна з ефективних форм його праці" [8]. Здібність мислити точно, формально, коли це потрібно, стає однією з важливих ознак загальної культури людини в сучасному високотехнологізованому світі. Ця здібність набуває вагомого значення при освоєнні сучасних професій, що пов’язуються з операторською діяльністю, якій властиве оперативне мислення.

Вивчаючи візуальне програмування можна відразу розвивати такий тип мислення, який знадобиться студентам і не тільки в програмуванні.

У обчислювальній техніці візуальна мова програмування(ВМП) - це будь-яка мова програмування, яка дозволяє користувачам створювати програми, маніпулюючи елементами програми графічно, а не вказуючи їх текстуально. ВМП дозволяє програмувати за допомогою візуальних виразів, просторових розташувань тексту і графічних символів, використовуваних або в якості елементів синтаксису, або у вторинній нотації.

Наприклад, багато візуальних мов програмування базується на ідеї «фігур і ліній», де фігури (прямокутники, овали та інші екранні об’єкти) розглядаються як суб’єкти і з’єднуються лініями (стрілками, дугами тощо), які являють собою відношення.

Візуальні мови програмування можуть бути додатково класифіковані, залежно від типу і ступеня використовуваного візуального вираження, на мови на основі значків, мови на основі форм і мови діаграм. Візуальні середовища програмування надають графічні або знакові елементи, якими користувачі можуть маніпулювати в інтерактивному режимі відповідно до певної просторової граматики для побудови програм.

Загальна мета ВМП - зробити програмування більш доступним для новачків і підтримати програмістів на трьох різних рівнях:

- Синтаксис: ВМП використовують значки/блоки, форми та діаграми, намагаючись зменшити або навіть повністю усунути потенційні синтаксичні помилки, допомагаючи з розташуванням примітивів програмування для створення добре сформованих програм. Наприклад, в не візуальних мовах програмування - здійснюється перевірка орфографії в текстових процесорах, що підкреслює або навіть автоматично виправляє окремі слова чи граматику.
- Семантика: ВМП можуть надати деякі механізми для розкриття значення примітивів програмування. Сюди можуть входити довідкові функції, що надають інформацію стосовно елементів, які вбудовані в мову програмування.
- Прагматика: ВМП підтримують вивчення практичних проблем програмування, в тому числі розуміння роботи програми в контексті конкретних ситуацій. Цей рівень підтримки дозволяє користувачам поміщати об’єкти, створені за допомогою ВМП, в певний стан, щоб дослідити, як програма буде реагувати на цей стан. Наприклад: в AgentSheets або AgentCubes користувачі можуть встановити ігри або симуляції в певний стан, щоб побачити, як програма буде реагувати. За допомогою мови програмування Thymio користувачі можуть привести робота в певний стан, щоб побачити, як він буде реагувати, тобто які датчики будуть активовані.[9]

Візуально перетворена мова - це невізуальна мова з накладеним візуальним поданням. Справжнім повноцінним візуальним мовам притаманне візуальне вираження, для якого немає очевидного текстового еквівалента. Тобто програму, яку створено у такій мові, може мати такий текстовий код, який не так явно відтворити у текстовій мові програмування.

Сучасні розробки намагаються інтегрувати підхід візуального програмування до мов програмування потоків даних(англ. dataflow

programming), щоб або мати негайний доступ до стану програми, що призводить до налагодження в режимі онлайн, або до автоматичної генерації програм і документування.

Синтаксичні аналізатори для візуальних мов програмування можуть бути реалізовані за допомогою графових граматик.

Середовище розробки ВМП часто поєднується зі спрощеним (або спеціалізованим) середовищем виконання, тому користувачі можуть швидко і легко запускати свою програму і бачити результати. Це не обов'язково безпосередньо пов'язано з самим ВМП, але це важливий фактор, що полегшує новачкам запуск їх першої програми, не турбуючись про технічні деталі.

Інший момент полягає в тому, що візуальне програмування може полегшити бачення загальної картини. Програма визначається її формою, тому як початківці, так і просунуті користувачі можуть отримати уявлення про те, що робить програма з першого погляду. Форми і кольори також використовують візуальні сприйняття користувача більшою мірою, ніж відступи і забарвлення коду, щоб зробити вихідний код більш читабельним. Це допомагає легше описати програму: ви можете представити її іншим і пояснити її основні моменти, навіть людям, які не знайомі з ВМП, за допомогою якої створена ця програма, показуючи графічний вихідний код безпосередньо замість малювання абстрактної діаграми; те, що ви показуєте, є джерелом безпосередньо. Це також допомагає ремонтопридатності: коли ви дивитеся на код ВМП, який вам не знайомий, ви можете легше побачити, що він робить, і як складні елементи збираються разом, щоб скласти програму. Блокова структура також означає, що часто буває легко перебудувати графічний код, на відміну від стомлюючого рефакторингу текстового вихідного коду.

Однак за межами деяких конкретних областей досі візуальні мови програмування далеко не так популярні, як класичні мови програмування.

Одна з причин, чому вони іноді сприймаються як розчарування, полягає в тому, що на відміну від намальованих від руки "прямокутників і стрілок",

вони все ще вимагають точного, однозначного визначення потоку управління. Іншими словами, візуальне програмування - це все ще програмування.

І хоча ВМП можуть полегшити навчання і зменшити синтаксичні труднощі, кожен тип програмування вимагає від користувачів знайомства з загальними, а також специфічними для мови концепціями програмування (наприклад, концепцією змінних і узагальненнями імперативного програмування). Таким чином, щоб мати можливість писати програму за допомогою ВМП, вам все одно потрібно думати як програміст.

Інші фактори можуть пояснити погану репутацію візуальних мов програмування у багатьох розробників.

Якщо ми подивимося на професійних розробників, які пишуть програми кожен день, візуальне програмування може здатися не дуже корисним. Коли користувач знайомий з синтаксисом мови програмування і повинен працювати з багатьма різнорідними компонентами або бібліотеками, переваги ВМП для виявлення синтаксису не дуже цікаві. Тоді як з точки зору представлення виразів текст є одночасно дуже компактним і відкритим: багато можливих слів можуть поміститися в просторі, який займе "блок" ВМП, і при цьому він буде все ще читабельним і запам’ятовуваним, і ви можете легко посилатися на зовнішні функції і розширювати мову, додаючи слова там, де в ВМП є тільки так багато легко впізнаваних форм і кольорів, які користувач може запам’ятати. Незважаючи на те, що деякі ВМП поєднують форми і текст, їх візуальні зображення зазвичай не можуть конкурувати з інформаційною щільністю тексту. Таким чином, незважаючи на те, що переваги візуальних мов з точки зору читабельності коду все ще зберігаються у розробників, їх переваги для доступності не є вирішальним фактором.

Крім того, середовища розробки ВМП іноді є спеціалізованими, застосовуваними до обмеженої області (напр. ігровий дизайн). Таким чином, програми можуть виконуватися безпосередньо в інтегрованих середовищах виконання, а зазвичай використовувані логічні блоки не настільки численні,

щоб виявлення синтаксису ставало скрутним. Але ці моменти важко досягти за допомогою ВМП загального призначення.

Ще одним фактором є те, що навіть деякі "серйозні" ВМП мали технічні недоліки, зокрема у швидкості їх виконання.

Мови візуального програмування можуть бути додатково класифіковані в залежності від типу і ступеня візуального вираження, на типи:

- мови на основі об’єктів, коли візуальне середовище програмування надає графічні або символьні елементи, якими можна маніпулювати інтерактивним чином відповідно до деяких правил;
- мови, в інтегрованому середовищі розробки яких на етапі проектування інтерфейсу застосовуються форми, з можливістю налаштування їх властивостей;

Користувачі часто плутають Microsoft Visual Studio, CodeGear Delphi і C++ Builder і мови, які включаються цей засіб (Visual C#, Visual J# ,Visual Basic) з мовами візуального програмування, проте ці мови є текстовими. Delphi та MS Visual Studio є візуальними середовищами програмування, але не візуальними мовами програмування.

До технології візуального програмування належить робота з середовищами, що використовують блоки команд, перетягуючи які за допомогою маніпулятора миші й виставляючи в потрібному порядку, можна створювати різні програми. Такими середовищами є Scratch, App Inventor, Google Blockly та ін. Ці середовища використовують візуальні мови програмування для розробки програм різного типу. Scratch призначений для створення ігор й анімацій; App Inventor - для написання програм для операційної системи Android; Google Blockly - для розробки ігор, для програмування контролерів Arduino й розробки інших програм. Розглянемо види візуальних мов програмування більш детально.

Види візуальних мов програмування

Scratch та подібні (рис. 1.6).

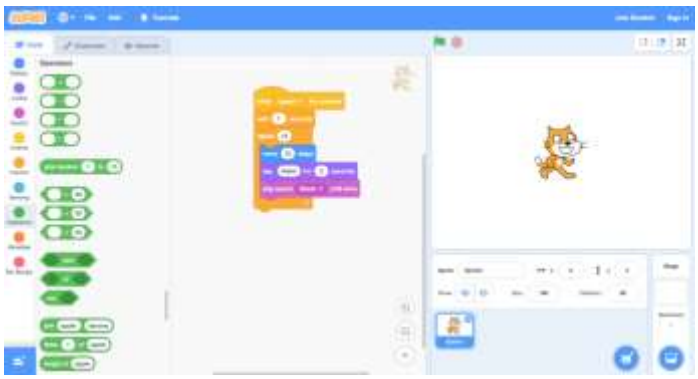

Рис. 1.6. ВМП «Scratch».

Scratch та його похідні - це, напевно, те, що більшість людей уявляє, коли вони думають про ВМП. Його граматика фактично є граматикою класичної імперативної мови програмування, але графічні підказки допомагають зрозуміти, як поєднувати основні елементи, такі як змінні, умови або регулятори потоку. Усередині блоків текст використовується для опису блоків, а текстові поля - для визначення імен і значень змінних. Можна розширити мову за допомогою настроюваних блоків, які також можуть бути визначені за допомогою Scratch. Крім візуальної граматики, сам оригінальний Scratch включає в себе середовище виконання з графічним рендерингом для легкого створення базових анімацій або ігор. До цього типу можна віднести і MIT App Inventor.

ВМП на основі блок-схем (рис. 1.7).

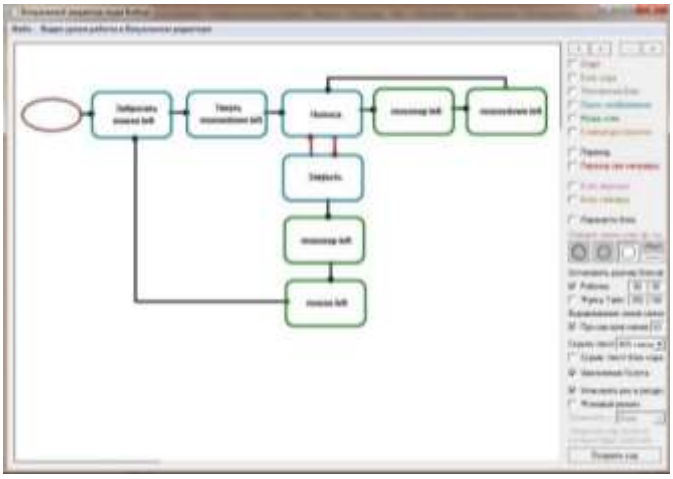

Рис. 1.1.7. ВМП «Кибор»

Виходячи безпосередньо з ідеї використання "прямокутників і стрілок" для опису програм, ці ВМП використовують візуальне представлення, близьке до блок-схем для опису основного потоку управління. Вони являють собою спрямовану послідовність виконання між блоками, з потоком, що проходить через блок до наступного, часто з розгалуженнями, які використовують результат/вихід блоку, щоб вибрати, який блок виконати наступним.

Цей фокус на виконанні та простій візуальній граматиці означає, що формат простий для розуміння, тому виявлення синтаксису не є проблемою. Але логічні конструкції, які можуть бути створені безпосередньо через такі ВМП, обмежені, багато що залежить від того, що знаходиться всередині блоків, що часто є заданим і не може бути змінено з графічного інтерфейсу.

Деякі формати, подібні блок-схемам, мають розширену граматику, яка дає можливість писати більш складні інструкції безпосередньо з графічного інтерфейсу.

Програмування потоку даних (рис. 1.8).

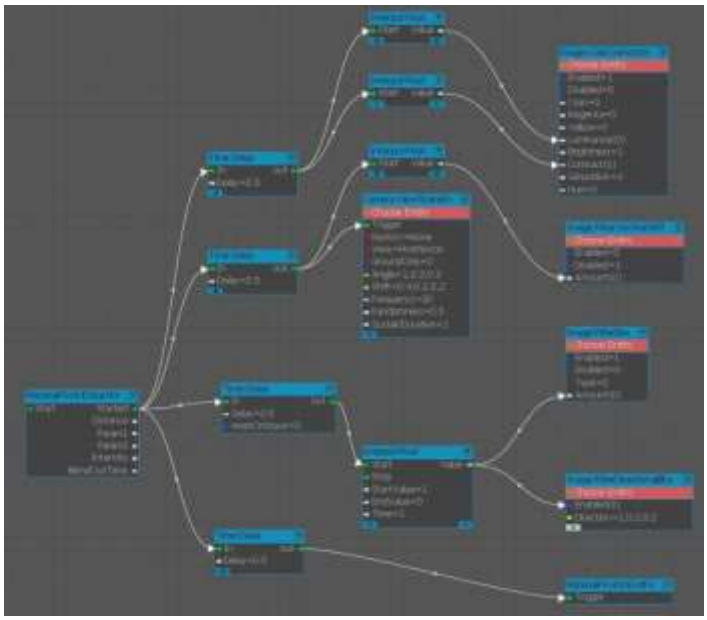

Рис. 1.8. ВМП «CryEngine flow graph»

Цей формат є одним з найбільш часто використовуваних ВМП для професійних додатків, орієнтованих на дизайнерів, а не на кінцевих користувачів або початківців програмістів.

При роботі з програмуванням потоку даних блоки являють собою функції, і вони пов’язані один з одним уздовж потоку даних, уздовж входу і виходу. Створюючи зв’язок між вихідним маркером одного блоку і вхідним маркером іншого блоку, користувач визначає потік виконання програми через потік даних.[10]

Представлено найбільш поширені та популярні типи, проте це не всі які існують.

Таким чином, візуальна граматика досить проста, користувач в основному вибирає, які блоки використовувати і як їх зв’язати. Це означає, що більша частина того, що визначає програму, насправді знаходиться всередині блоків, які зазвичай пишуться на класичній мові програмування.

Використання таких середовищ ілюструє розуміння поняття «візуальне програмування» як способу створення програм за допомогою блоків, зіставляючи які, можна скласти набір інструкцій. Для цього використовується як традиційний (цикл, умовні оператори), так і додатковий набір керівних конструкцій (рух, обертання, малювання, програмування звуку тощо), процедури й функції з параметрами, повноцінний набір логічних виразів, а також можливості роботи з кольором і графікою. Користувачам надаються тільки елементарні можливості введення з клавіатури й виведення окремих виразів у вікні. Для таких середовищ характерний сильний акцент на візуальному складникові, що компенсується в процесі професійної роботи в деяких з них можливістю компіляції на традиційну мову програмування за вибором.

У міру розширення комп’ютерів і того, як збільшується потужність комп’ютерного обладнання, збільшується і застосування візуальних мов програмування. У той час як комп’ютери отримують розробки програмного забезпечення, які можуть обробляти ці мови програмування, люди часто занадто спеціалізовані в конкретній мові програмування, щоб успішно використовувати візуальне програмування.

Останнім часом набирає популярність ніша напівкодових або низькокодових програмних платформ - і навіть Amazon хоче приєднатися до розробки в цьому сегменті[11]. Ці підходи беруть складність текстових мов і об’єднують їх з візуальною графікою. Це дозволяє користувачам реалізовувати графіку для взаємодії з кодом, відкриваючи відносини між користувачем і кодом на абсолютно новому рівні. Поєднання майбутніх розробок програмного забезпечення та ВМП забезпечує найкраще з обох світів.

1.2 Середовище MIT App Inventor 2

Смартфон є інформаційною сполучною ланкою в сучасному цифровому столітті, з доступом до майже нескінченної кількості контенту в Інтернеті, в поєднанні з багатьма датчиками і особистими даними. Однак людям важко використовувати всю потужність цих поширених пристроїв для себе і свого оточення. Більшість користувачів смартфонів користуються технологією, не будучи в змозі розробити її самостійно, хоча локальні проблеми часто можуть бути вирішені за допомогою мобільних пристроїв. Яким чином вони тоді можуть навчитися використовувати можливості смартфонів для вирішення реальних, повсякденних проблем? MIT App Inventor розроблений для популяризації цієї технології і використовується в якості інструменту для навчання обчислювального мислення в різних освітніх контекстах, навчаючи людей створювати додатки для вирішення проблем у своїх колах інтересу.

У цьому пункті буде описано цілі MIT App Inventor і те, як вони вплинули на наш дизайн і розвиток - від початку програми в Google в 2008 році, через міграцію в MIT, до сьогоднішнього дня. Також обговоримо педагогічну цінність MIT App Inventor і його використання в якості інструменту для навчання і заохочення студентів думати і діяти за алгоритмом.

MIT App Inventor - це середовище розробки веб-додатків, спочатку створене компанією Google, а тепер підтримується Массачусетським технологічним інститутом (Massachusetts Institute of Technology). Він дозволяє новачкам в комп’ютерному програмуванні створювати прикладне програмне забезпечення (додатки) для Android. Це безкоштовне програмне забезпечення з відкритим кодом.

Він використовує графічний інтерфейс користувача (GUI), дуже схожий на мови програмування Scratch і StarLogo, який дозволяє користувачам перетягувати візуальні об’єкти для створення програми, яка може працювати на пристроях Android. Створюючи App Inventor, компанія Google спиралася на

значні минулі дослідження в галузі комп'ютерної освіти та досвід, здобутий у створенні Google середовищ онлайн розробки. [12]

App Inventor та інші проекти засновані на конструкціоністських теоріях навчання, які підкреслюють, що програмування може бути засобом залучення потужних ідей через активне навчання. Як така, вона є частиною постійного руху в області комп'ютерів і освіти, яке почалося з роботи Сеймура Паперта і групи логотипів MIT в 1960-х роках, а також проявилося в роботі Мітчелла Резніка над Lego Mindstorms і StarLogo. [13]

App Inventor також підтримує використання хмарних даних за допомогою експериментального компонента бази даних Firebase # Firebase Realtime.

Історія створення. Заявка для створення середовища було подано у липні 2010 року і опублікована в грудні цього ж року. Команда розробників App Inventor очолювалася Хелом Абельсоном і Марком Фрідманом. У другій половині 2011 року Google випустила вихідний код, припинила роботу свого сервера і надала фінансування для створення центру мобільного навчання Массачусетському технологічному інституту, очолюваного творцем App Inventor Хелом Абельсоном та іншими професорами MTI Еріком Клопфером і Мітчелом Резніком. Версія MTI була запущена в березні 2012 року.

- У грудні 2013 року (початок Години коду) MTI випустив App Inventor 2, перейменувавши оригінальну версію «App Inventor Classic». Основні відмінності полягають в тому, що редактор блоків в оригінальній версії виконувався в окремому Java-процесі, використовуючи бібліотеку Open Blocks Java для створення візуальних блоків мов програмування та програмування
- Open Blocks поширюються програмою Scheller Teacher Education Program (STEP) Массачусетського технологічного інституту і є похідними від магістерських дисертаційних досліджень Рікарози Роке. Професор Ерік Клопфер і Даніель Вендель з програми Шеллера підтримали поширення Open Blocks під ліцензією MTI. Візуальне

програмування відкритих блоків тісно пов’язане з StarLogo TNG, проектом STEP, і Scratch, проектом MTI Media Lab Lifelong Kindergarten Group на чолі з Мітчелом Резніком.

App Inventor 2 замінив Open Blocks на Blockly, редактор блоків, який працює у веб-браузері.

Інтерфейс середовища. Користувальницький інтерфейс MIT App Inventor включає в себе два основних редактора: редактор дизайну і редактор блоків. Редактор дизайну, або конструктор (рис. 1.9), являє собою інтерфейс перетягування для розміщення елементів користувальницького інтерфейсу програми (User Interface).

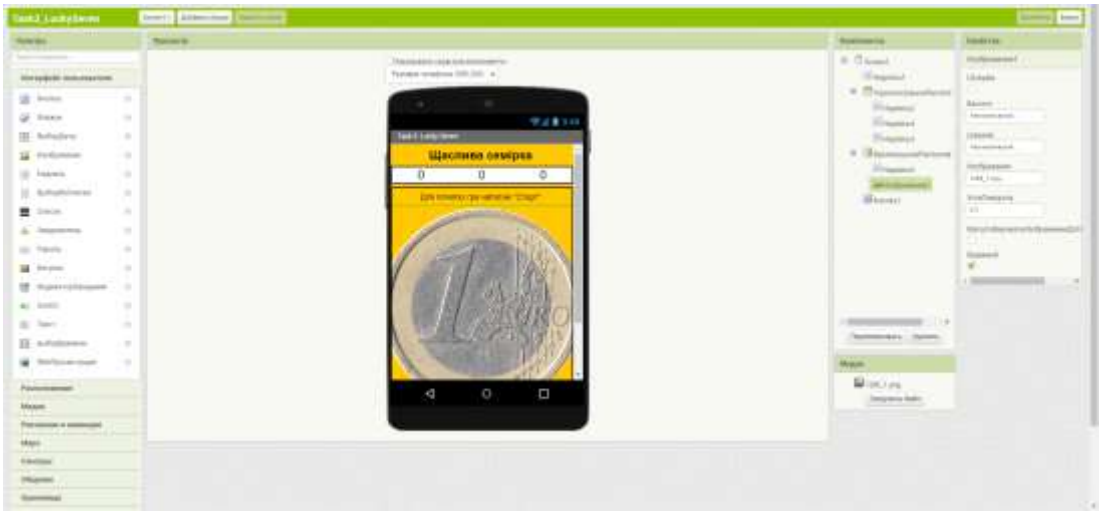

Рис. 1.9. Редактор дизайну MIT App Inventor.

Редактор дизайну можна умовно поділити на чотири частини: «палітра» - в ній знаходяться всі можливі компоненти, які можна використовувати для створення додатку; вікно перегляду програми - тут ми бачимо як виглядає сама програма на екрані пристрою; «компоненти» - ця частина відображає всі компоненти, які були задіяні для створення додатку; «властивості» - в даній частині можна побачити властивості компоненту на який було натиснуто, тут їх можна редагувати.

Редактор блоків (рис. 1.10) - це середовище, в якому розробники додатків можуть візуально викладати логіку своїх додатків, використовуючи

кольорові блоки, які з’єднуються разом, як шматочки головоломки, щоб описати програму.

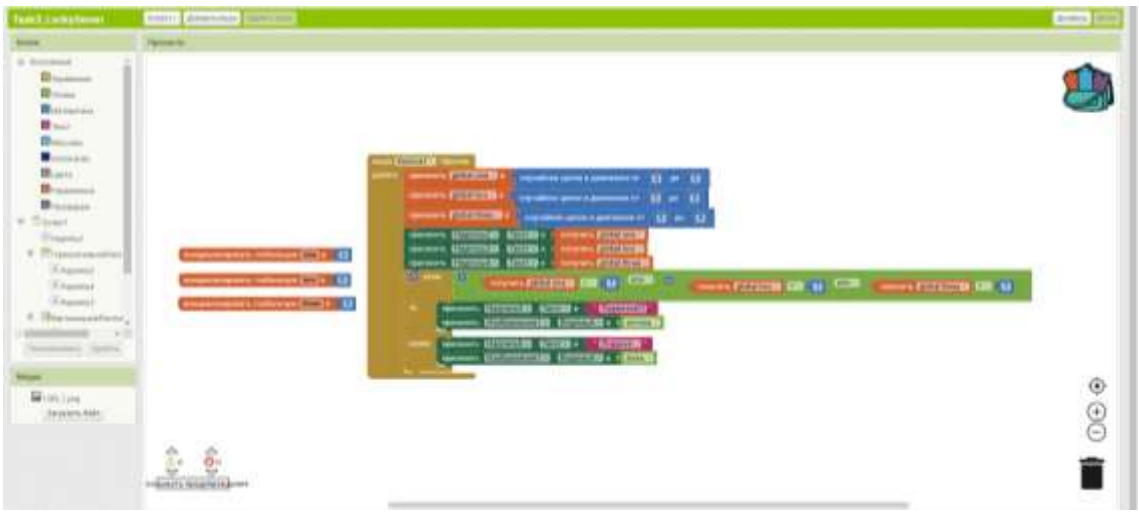

Рис. 1.10. Редактор блоків MIT App Inventor.

Редактор блоків, в свою чергу, можна поділити на дві частини: «блоки» - в цій частині знаходиться вкладка в якій містяться блоки, окрім загальних блоків, є блоки до компонентів, які використані в додатку; вікно перегляду та конструювання блоків - в ньому власне і будуються блоки команд.

Щоб допомогти в розробці і тестуванні, App Inventor надає мобільний додаток під назвою App Inventor Companion (або просто «Компаньйон»), який розробники можуть використовувати для тестування і налаштування поведінки своїх додатків в режимі реального часу. Його можна безкоштовно завантажити в магазині додатків на смартфоні. Таким чином, будь-який бажаючий може швидко створити мобільний додаток і відразу ж приступити до ітерації і тестування.

При розробці MIT App Inventor головною метою було впровадження розробки мобільних додатків в освітніх цілях. До його випуску більшість середовищ розробки для мобільних додатків були складними, доступними тільки для фахівців в області системного рівня або вбудованого програмування, або всього разом. Навіть з операційною системою Google Android і мовою програмування Java розробка користувальницького інтерфейсу була складним завданням. Крім того, для успішного використання

платформи було потрібно знання синтаксису і семантики Java, а також вміння налагоджувати помилки компіляції Java (наприклад, неправильно написані змінні або недоречні точки з комою). Ці проблеми представляли собою бар'єри для входу людей, які не обізнані в комп'ютерних науках, що є цільовою аудиторією App Inventor.

Компоненти (або елементи) - це основні абстракції в MIT App Inventor. Компоненти зменшують складність управління взаємодіями зі специфічними для платформи інтерфейсами прикладного програмування (Application Programming Interfaces) і деталями, що стосуються управління станом апаратного забезпечення пристрою. Це дозволяє користувачеві думати про поточну проблему, а не про дрібниці, які зазвичай потрібні розробникам додатків. Наприклад, той, хто планує використовувати MIT App Inventor для створення програми, що використовує глобальну систему позиціонування (GPS) для відстеження руху, не повинен займатися управлінням життєвим циклом програми, програмними і апаратними блокуванням GPS або мережевим підключенням (у випадку, якщо виявлення місця розташування повертається до мережевого розташування). Замість цього розробник програми додає компонент датчика місцезнаходження, який абстрагує цю складність і надає API для включення і обробки оновлень місця розташування. Більш конкретно, ця реалізація скорочує 629 рядків коду Java до 23 блоків, з яких тільки два потрібні для виконання відстеження місця розташування. Це зниження складності дозволяє розробникам додатків зосередитися на поточній проблемі і швидко досягти поставленої мети. [14]

Компоненти складаються з трьох типів блоків: властивостей, методів і подій. Властивості керують станом компонента і доступні для читання та/або запису розробником програми. Наприклад, властивість enabled датчика розташування включає в себе функціональність, необхідну для налаштування GPS-приймача і управління його станом під час використання програми. Методи працюють з декількома параметрами і можуть повертати результат. Події реагують на зміни стану пристрою або програми в залежності від

зовнішніх факторів. Наприклад, коли користувач змінює своє місце розташування, подія, яка відповідає за зміну положення надає логікі додатку команду реагувати на зміни.

У MIT App Inventor, користувачі кодують поведінку та роботу програми за допомогою мови програмування основаної на блоках. App Inventor має два типи блоків: вбудовані блоки та компонентні блоки. Вбудована бібліотека блоків надає базові функції та операції, які зазвичай доступні в інших мовах програмування, такі як логічні значення, рядки, числа, списки, математичні оператори, оператори порівняння та оператори потоку управління. Розробники використовують блоки компонентів (властивості, методи і події) для реагування на системні і призначені для користувача події, взаємодії з апаратними засобами пристроїв і налаштування візуальних і поведінкових аспектів компонентів.

Вся логіка програми побудована на трьох типах блоків верхнього рівня: визначення глобальних змінних, визначення процедур і обробники подій компонентів. Глобальні змінні надають іменовані слоти для зберігання станів програми. Процедури визначають загальну поведінку, яка може бути викликана з декількох місць в коді. Коли на пристрої відбувається подія, вона запускає відповідну поведінку програми, запропоновану в блоці подій. Блок обробника подій може посилатися на глобальні змінні або процедури.

Команда розробників App Inventor при проектуванні середовища враховувала ряд обмежень. Пропонується розглянути кілька проектних рішень, їх обґрунтування і вплив на обчислювальне мислення в App Inventor:

- отримуєш те, що бачиш (What You See Is What You Get): редактор дизайну для App Inventor дозволяє розробникам бачити, як додаток буде відображатися на екрані пристрою, і налаштовувати форм-фактор візуалізованого пристрою (наприклад, телефону або планшета). Коригування властивостей візуальних компонентів, наприклад, кольору і розміру фону, відображаються в реальному часі. Додатки також можуть

бути запущені в режимі живої розробки за допомогою «Компаньйона», про який більш детально поговоримо нижче.

- на відміну від багатьох мов програмування, App Inventor обмежує створення нових сутностей під час виконання. Це дає велику кількість переваг. По-перше, явно розташовуючи всі компоненти в додатку, користувач може візуалізувати його чітко, а не міркувати про речі, які не будуть існувати до певного часу. По-друге, це зменшує шанси користувачів вводити циклічні залежності пам'яті в користувальницькому інтерфейсі, що в кінцевому підсумку призведе до того, що у програми закінчиться пам'ять. Це спонукає розробників додатків замислитися над тим, як правильно структурувати свої програми та повторно використовувати компоненти, щоб уникнути перевантаження системи або кінцевого користувача додатку.
- натуральне числення: система числення в App Inventor передбачає початкове значення 1. На відміну від більшості мов програмування, які більш узгоджені з архітектурою машини і тому починаються з 0.

Швидка ітерація та проектування з використанням «Компаньйона». Ключовою особливістю MIT App Inventor є його живе середовище розробки для мобільних додатків. App Inventor забезпечує це за допомогою супутнього додатка, встановленого на мобільному пристрої користувача. Веб-інтерфейс App Inventor відправляє код в супутній додаток, яке інтерпретує його і відображає розробнику в режимі реального часу(рис. 1.11). Таким чином, користувач може змінювати інтерфейс і поведінку програми в режимі реального часу. Наприклад, учень, який розробляє гру з м'ячем, в якій м'яч може відскочити від краю ігрового майданчика. Однак початкова версія додатку може не враховувати того, що м'яч зіткнеться зі стіною, і тому він зупиниться. Тестуючи додаток і коригуючи його програмування у відповідь на небажану поведінку, студенти можуть досліджувати його більш вільно.

Рис. 1.11. Вигляд «MIT Companion»

На зображені показано вигляд самого MIT Companion (зліва). Після того, як відбулося з'єднання з середовищем то додаток завантажується в «Компаньйона» і ми бачимо його вже на екрані пристрою та можемо взаємодіяти з ним (справа).

Традиційний цикл збірки для Android-програми включає в себе написання коду в текстовому редакторі або інтегрованому середовищі розробки, і перебудова програми для тестування часто може зайняти кілька хвилин, в той час як внесення змін в живе середовище розробки зазвичай відбувається за декілька секунд. Можна швидко побачити зміни, які відображені в додатку і це означає, що студенти можуть досліджувати, робити помилки під час вивчення, але швидко їх знаходити та виправляти. Це покращує та спрощує вивчення.

Основна мета MIT App Inventor - надати всім, хто зацікавлений у створенні додатків, необхідні для цього інструменти. Навчальні матеріали, розроблені командою розробників, в першу чергу орієнтовані на вчителів та

учнів середніх, старших класів та студентів, але створити додаток зможе кожен, якщо цього забажає.

Розміщуючи результати студентського програмування на мобільних пристроях, App Inventor дозволяє студентам перенести свою роботу із робочого місця за комп'ютером в повсякденне життя і оточення. Цей перехід впливає на те, що студенти створюють і як вони представляють себе в якості цифрових творців. Це дозволяє студентам перенести своє уявлення про себе з людей, які «вміють кодувати», на членів суспільства, наділених можливостями надавати реальний вплив на їхнє життя і життя інших людей.

Отже, представлено багато аспектів програми MIT App Inventor з точки зору розвитку та освіти. Є деякі помилки, обмеження та переваги, які важливо висвітлити.

Одна з поширених позицій противників App Inventor полягає в тому, що програмування за допомогою блоків не є реальним програмуванням (часто порівнюють мови блочного програмування з текстовими мовами). Це хибна дихотомія, якщо розуміти програмування як акт опису комп'ютеру певної реалізації машини Тьюрінга. Проте більшість користувачів усвідомлюють, що за допомогою таких інструментів, як App Inventor, вони можуть досягти реальних змін у своєму оточенні. Нові користувачі, які починають вивчати програмування з блочних мов, також схильні вивчати програмування далі і продовжувати частіше програмувати, ніж ті, що вивчають текстові мови.[15]

Ще одна поширена помилка полягає в тому, що створення мобільних додатків - це те, що можуть робити тільки експерти і ті, хто має великий досвід у програмуванні. Однак студенти використовують App Inventor для розробки власних мобільних додатків практично без попереднього досвіду. Для того, щоб почати працювати в цьому середовищі не обов'язково володіти хоча б однією текстовою мовою програмування.

Одне з обмежень мов програмування за допомогою блоків може бути те, що для студентів, які навчаються в блочних мовах, може бути важко перейти до текстового уявлення. Тому таким студенти, бажано, за можливістю,

допомогти перейти до текстових мов, намагаючись при цьому не втратити знання, що були отримані візуальною мовою. Професор Дейв Уолбер і команда Університету Сан-Франциско активно займаються цим питанням, розробляючи App Inventor Java Bridge[16]. Java Bridge дозволяє будь-якому користувачеві перевести додаток App Inventor в додаток Java, сумісний з Android Studio, офіційним текстовим середовищем розробки, яке використовується для створення додатків Android.

Ще одним обмеженням App Inventor є те, що його дизайн перешкоджає повторному використанню коду. Повторне використання коду є однією з ключових концепцій обчислювального мислення в фреймворку Бреннана і Резніка [17]. Багато текстових мов забезпечують надійну підтримку бібліотек коду та управління залежностями, що дозволяє розробникам додатків легше спиратися на роботу один одного. Хоча App Inventor надає галерею для публікації завершеного вихідного коду програми, спільноті ще належить покращити деталізацію бібліотек, які поширені в інших мовах програмування. Це дає можливість продовжувати розвивати платформу і спільноту користувачів та є гідним предметом для подальшого вивчення.

Тепер про переваги візуального програмування для мобільних пристроїв. Користувачі платформи App Inventor отримують користь від можливості перепрофілювати навички обчислювального мислення, які вони вивчають, щоб взаємодіяти з фізичним простором у зовнішньому світі. Візуальне програмування в App Inventor, а також абстракція і поділ концепцій на компоненти і блоки дозволяють розробнику додатку більше зосередитися на розкладанні своїх проблем на вирішувані елементи. Можливість запуску додатків на мобільних пристроях дозволяє студентам сприймати свої власні програми як частину екосистеми, з якою вони щодня взаємодіють і з якою вони добре знайомі. Оскільки така інкапсуляція скорочує час, необхідний для створення додатка, навіть простого прототипу, розробники додатків можуть швидко схоплювати суть і повторювати, не витрачаючи багато часу з точки

зору циклу компіляції-завантаження-запуску, який типовий для розробки мобільних додатків.

Висновки про MIT App Inventor. Проект MIT App Inventor продовжує розширювати межі освіти в контексті розробки мобільних додатків. Його абстрагування апаратних можливостей і зведення складної логіки до компактних уявлень дозволяє користувачам швидко і ефективно розробляти проекти. Ми обговорили, що навчання в середовищі візуального програмування App Inventor розвиває елементи обчислювального мислення. Окрім цього робота в таких середовищах розвиває також алгоритмічний тип мислення.

Тому можна підсумувати, що вивчення цього середовища буде корисним студентам, які хочуть і далі розвиватися в програмуванні, так і тим, хто не захоче продовжувати роботу в цій сфері.

РОЗДІЛ ІІ
МЕТОДИКА НАВЧАННЯ

2.1 Методична система навчання візуальному програмуванню

2.1.1 Мета, цілі курсу

На початку вивчення курсу потрібно пояснити та зацікавити студентів вивчати дану тему. В сучасному суспільстві люди, які вміють програмувати на високому рівні, можуть досить легко реалізуватися в цій сфері. Зараз епоха цифрових технологій і люди, які вміють створювати програмне забезпечення дуже ціняться. З огляду на це, можна зрозуміти, що в цій сфері знайти роботу з високою заробітною платою не складно. Якщо ти дійсно вправний програміст, тестер чи дизайнер комп'ютерних або мобільних додатків, то IT компанії самі будуть тебе шукати та пропонувати роботу з гарною заробітною платою та умовами.

Звичайно студенти, які прийшли навчатися на цю спеціальність, вже це розуміють, тому нагадавши їм це, вони лише впевняться в тому, що зробили правильний вибір обравши саме цю сферу. Це буде додатковою мотивацією до вивчення.

Оскільки сфер у програмуванні немало, є різні напрями, мови програмування - потрібно повідомити, що темою вивчення цього курсу стане візуальне програмування. Пояснити його особливості, сфери використання і звичайно розповісти про середовище, в якому будуть працювати студенти.

Мета навчання курсу: навчити студентів створювати програми самостійно для операційної системи Android; зацікавити програмуванню; ознайомити з принципами роботи у візуальних середовищах програмування, поняттям візуального програмування; пояснити актуальність вивченню програмуванню.

Цілі курсу: студенти повинні опанувати розробку додатків для ОС Android за допомогою візуального середовища програмування MIT AppInventor.

Після вивчення курсу студент повинен розуміти принцип роботи в MIT AppInventor, володіти інструментами для розробки, знати їх функції та призначення.

На закінченні вивчення курсу студент має розробити власний додаток в середовищі MIT AppInventor, продемонструвати його програмну складову(блоки команд), те, як він працює, мету цього додатку(навчальна, розважальна тощо) і як його можна використовувати.

2.1.2 Зміст

На вивчення теми відводилося 3 лекції та 9 лабораторних робіт. Перша лекція виступає введенням студентів з такими поняттями як: візуальне програмування, мови візуального програмування, їх приклади, App Inventor. Тобто перша лекція описує тему курсу, середовище в якому будуть працювати студенти далі.

Перелік тем та назв лабораторних робіт:

1. Робота з циклами, додаток «Факторіал».
2. Компоненти «Часы», «Звук», «Уведомитель», «Pedometr»; додатки «Таймер» та «Крокомір».
3. Робота з декількома екранами, додатки «Словник» та «Paint 0,5».
4. Створення додатку-гри «Відбивайка».
5. Створення додатку-гри «Відбивайка».
6. Робота над власними додатками.
7. Робота над власними додатками.
8. Робота над власними додатками.
9. Демонстрація створених додатків, залік.

План лекції:

I. **Організаційний момент**
II. **Оголошення теми, завдань лекції та курсу**
III. **Вивчення нового матеріалу**
IV. **Підведення підсумків лекції**

вид лекції: демонстраційна(пояснення з використанням проектору), до кожного слайду презентації йде пояснення виклад матеріалу лекції.

Слайд 1

Привітання, оголошення теми.

Слайд 2

Візуальне програмування - це вид програмування, що передбачає створення програм за допомогою наглядних засобів, тобто шляхом оперування графічними об’єктами, а не написання програмного коду в текстовому вигляді. Візуальне програмування часто представляють як наступний етап розвитку текстових мов програмування. Основною суттю візуального програмування є побудова розв’язку поставленої задачі за допомогою візуальних заготовок, які вставляються у форму, присвоєння значень їхнім атрибутам і створення чи застосування потрібних для розв’язання даної задачі методів. Останнім часом візуальному програмуванню стали приділяти більше уваги - у зв'язку з розвитком мобільних сенсорних пристроїв (смартфони, планшети). Візуальне програмування в основному використовується для створення програм з графічним інтерфейсом для операційних систем з графічним інтерфейсом користувача.

Слайд 3

Візуальне програмування виникло на основі об'єктно-орієнтованого програмування, як засіб автоматизації його процесів. Тепер для складання програми користувачу необхідно маніпулювати наданими графічними засобами - компонентами. Компоненти мають певні атрибути (властивості). Властивості можуть набувати значення зі заздалегідь фіксованого значення чи набору, або можуть бути придумані користувачем. Користувач розв’язує різноманітні задачі шляхом добирання компонентів надання потрібних значень їхнім атрибутам. При візуальному програмуванні програміст показує, що необхідно отримати в результаті, а текст програми генерується автоматично за допомогою візуального прототипу. Результати виводять на форму (вікно, характерне для операційної системи), де можна застосувати

різноманітні елементи керування, властиві для діалогових вікон прикладних програм (кнопки, різноманітні поля).

Значна частина візуальних мов програмування базується на ідеї «фігур і ліній», де фігури (прямокутники, овали та ін.) розглядаються як суб'єкти і з'єднуються лініями (стрілками, дугами тощо), які являють собою відношення.

Слайд 4

Мови візуального програмування можуть бути додатково класифіковані в залежності від типу і ступеня візуального вираження, на типи:

- мови на основі об'єктів, коли візуальне середовище програмування надає графічні або символьні елементи, якими можна маніпулювати інтерактивним чином відповідно до деяких правил;
- мови, в інтегрованому середовищі розробки яких на етапі проектування інтерфейсу застосовуються форми, з можливістю налаштування їх властивостей;

Слайд 5

До технології візуального програмування належить робота з середовищами, що використовують блоки команд, перетягуючи які за допомогою маніпулятора миші й виставляючи в потрібному порядку, можна створювати різні програми. Такими середовищами є Scratch, App Inventor, Google Blockly та ін. Ці середовища використовують візуальні мови програмування для розробки програм різного типу. Scratch призначений для створення ігор й анімацій; App Inventor - для написання програм для операційної системи Android; Google Blockly - для розробки ігор, для програмування контролерів Arduino й розробки інших програм.

Використання таких середовищ ілюструє розуміння поняття «візуальне програмування» як способу створення програм за допомогою блоків, зіставляючи які, можна скласти набір інструкцій.

Слайд 6

Так як за допомогою App Inventor можна створювати додатки для Андроїд смартфонів, трішки поговоримо про сам Андроїд. Як думаєте, чому саме для Андроїд?

Найпопулярніша ОС для смартфонів. Незважаючи на початкову заборону на установку програм з «неперевірених джерел» (наприклад, з карти пам'яті), це обмеження відключається штатними засобами в налаштуваннях пристрою, що дозволяє встановлювати програми на телефони та планшети без інтернет-підключення а також дозволяє будь-кому безкоштовно писати програми для Android і тестувати на своєму пристрої.

Слайд 7

App Inventor — середовище візуальної розробки android-застосунків, що вимагає від користувача мінімальних знань програмування. Для програмування в App Inventor використовується графічний інтерфейс користувача, візуальна мова програмування дуже схожа на мову Scratch. Для роботи в MIT App Inventor необхідно мати обліковий запис Google.

Після завершення презентації та викладу основного матеріалу, викладач переходить на сайт середовища MIT App Inventor та описує вигляд сайту, інтерфейс самого середовища. Приклад план-опису:

Інтерфейс App Inventor 2

1. Сайт http://appinventor.mit.edu/
2. Заходимо у свій аккаунт, відразу бачимо список проектів.
3. Проект, ключ розробника гугл для публікації в Google Play, види підключення, «построить»,
4. Екран розробки: поділений на частини. Два вікна розробки: дизайн, блоки.
5. Вибір екранів. Центр екрану - Просмотр.
6. Палітра - перелік об’єктів і компонентів, які можна добавити в додаток. Знак питання - справка по об’єкту. Перегляд вкладок. Програмування роботів Лего.

7. Компоненти - перелік всіх об’єктів і компонентів, які використовуються в додатку.

8. Медіа - всі об’єкти медіа(відео, фото, звуки), які завантажено.

9. Властивості.

10. Блоки, корзина, рюкзак, попередження, помилки.

11. Запуск додатку(qr код, apk файл), MIT AI2 Companion.

12. Емулятор

13. Новий проект(hello world), додавання компонентів, свойства екрану, компонентів. Настройка властивостей екрану, надпису. Перегляд в емуляторі. Hello world. Додаємо елемент «Кнопка». Настройка властивостей кнопки.

14. Переходимо в Блоки. Команди для кнопки, тексту. Запуск емулятору, перевірка роботи додатку.

Після опису інтерфейсу, на прикладі, показується робота в середовищі. Викладач створює додаток, а студенти спостерігають процес на проекторі. При створенні додатку, потрібно пояснювати кожен крок, і при використанні інструментів додатково проговорювати їх властивості, функції.

На першій лекції пропонується до розгляду створення елементарної програми: за натисканням кнопки буде відбуватися переклад тексту з англійської мови на українську (рис. 2.1, 2.2).

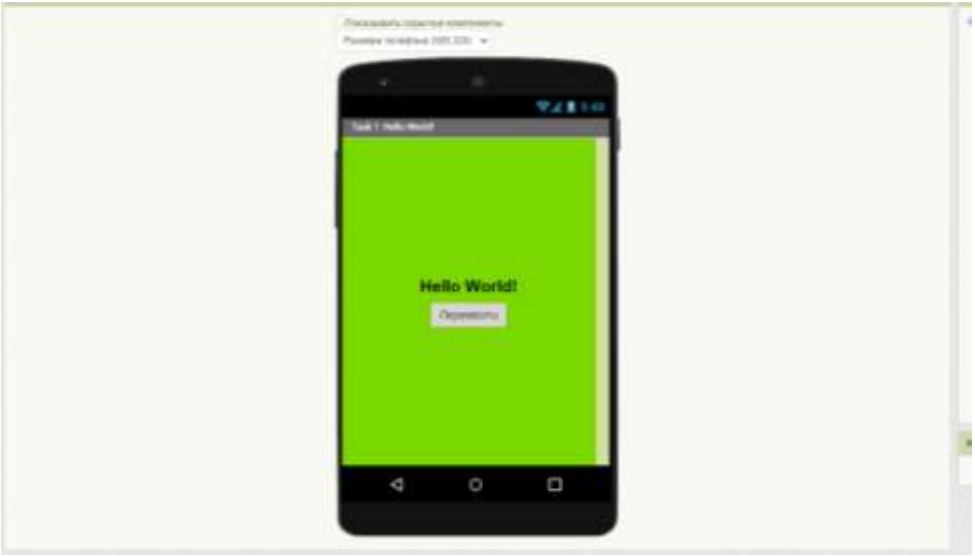

Рис. 2.1. Вигляд програми на екрані смартфону.

Рис. 2.2. Блок команд програми.

Після створення програми за допомогою емулятора демонструється його робота. На цьому перша лекція завершується.

Друга лекція продовжує те, на чому завершилася перша. А саме продовжується пояснення принципу роботи у середовищі MIT AppInventor на прикладі створення додатків.

План лекції

I. **Організаційний момент**
II. **Оголошення теми лекції**
III. **Вивчення нового матеріалу**
IV. **Підведення підсумків лекції**

вид лекції: демонстраційна(пояснення з використанням проектору)

На початку лекції проводиться невелике опитування стосовно теми курсу, що вивчається, для нагадування. Друга лекція має на меті пояснити більш детально практичну роботу в MIT AppInventor, тому студенти спостерігатимуть за створенням додатку. На цій лекції буде розглянуто два додатки «Калькулятор» та гра «Щаслива Сімка». Першою розглядається програма «Калькулятор»(рис. 2.3, 2.4):

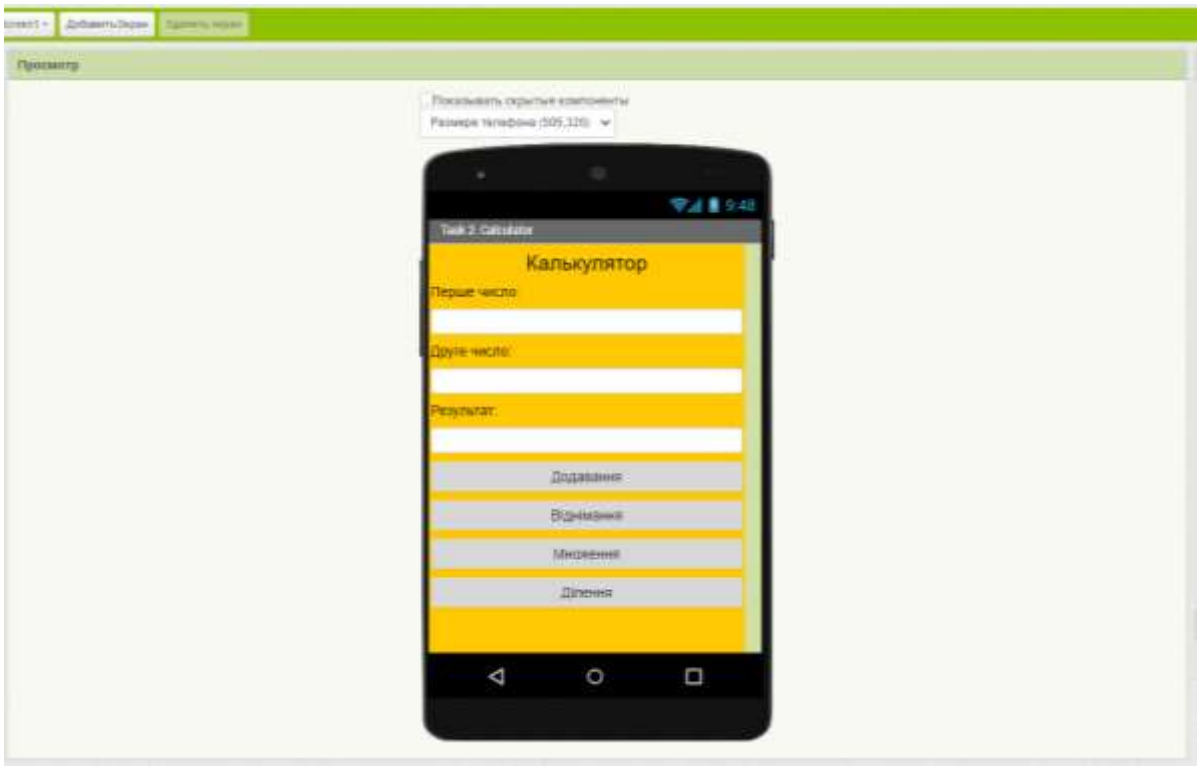

Рис. 2.3. Вигляд програми «Калькулятор».

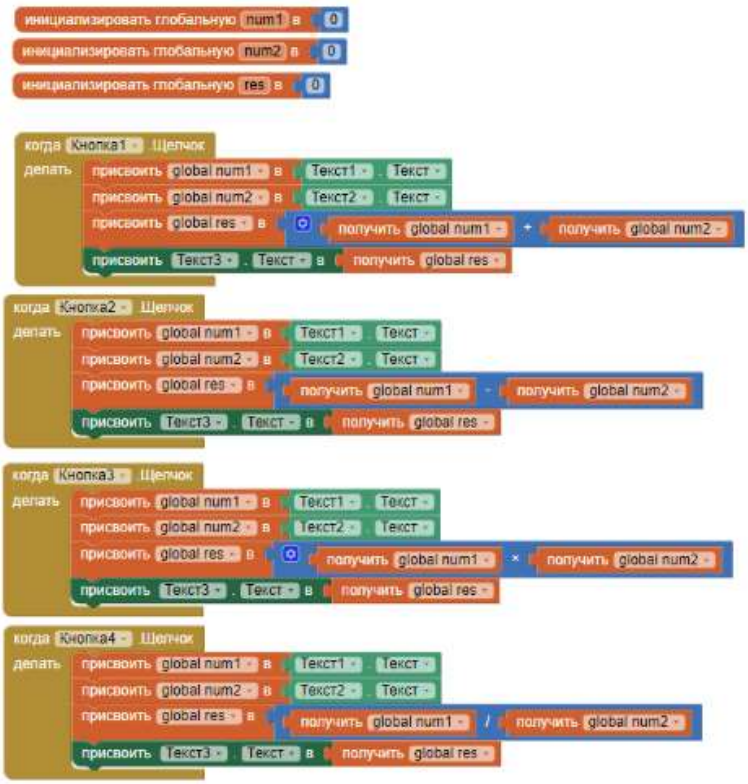

Рис. 2.4. Блоки команд «Калькулятор».

Демонстрація створення цієї програми потрібне для того, щоб пояснити як працювати з найпоширенішими елементами середовища, їх створювати та налаштовувати.

Далі гра «Щаслива Сімка»(рис. 2.5, 2.6):

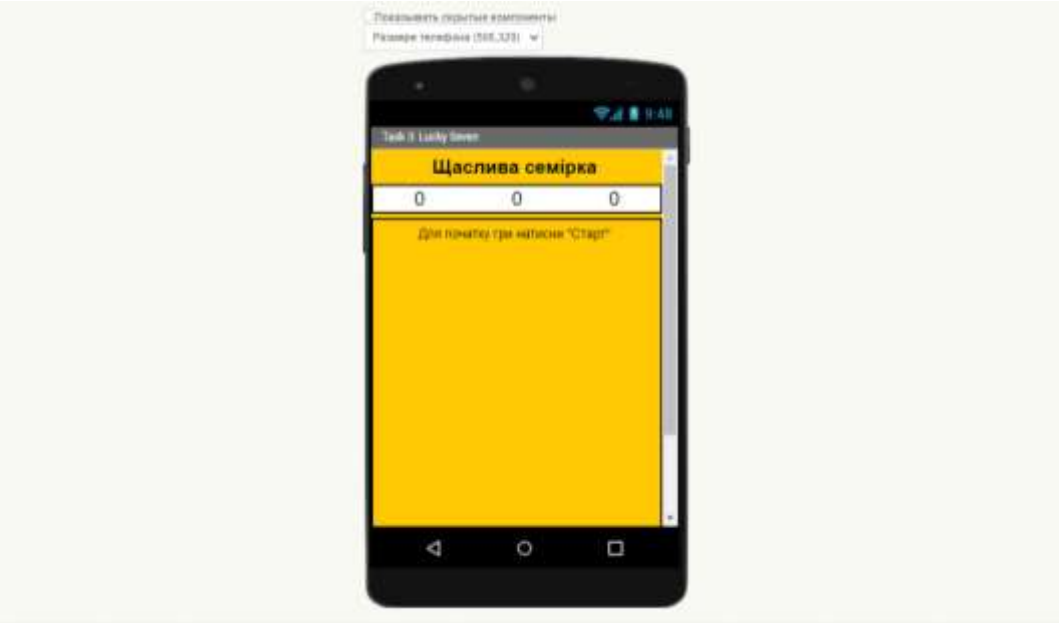

Рис. 2.5. Вигляд гри «Щаслива Сімка».

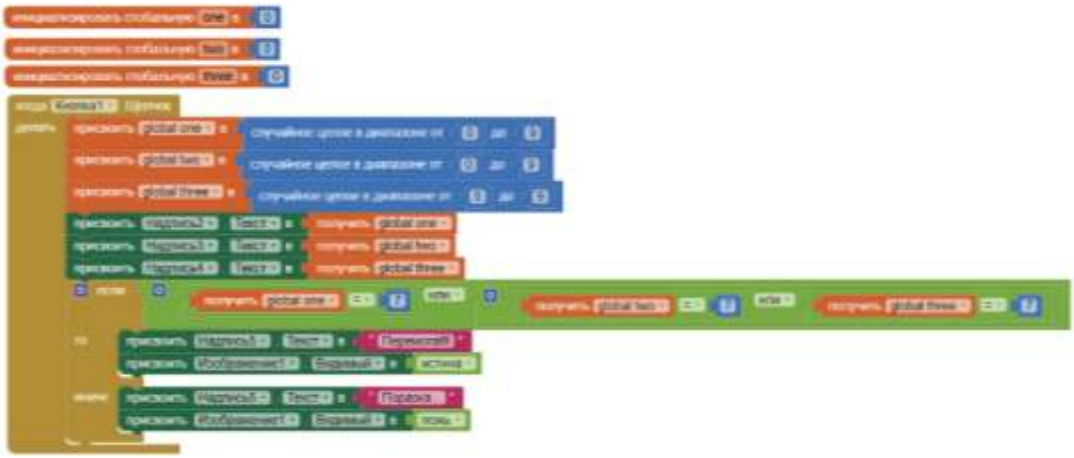

Рис. 2.6. Блоки команд «Щасливої Сімки».

Цей додаток розвиває вміння працювати з розгалуженнями, перевіркою умови. А для цього пояснюється як працювати з відповідними елементами.

При створенні програм, звичайно, пояснюється властивості об’єктів, які використовуються, їх функції.

На закінченні лекції підводяться підсумки вивченого: які інструменти вивчили, які в них функції, властивості, для яких задач їх можна

використовувати. Проте викладання такого типу матеріалу варто завершувати практичним закріпленням, тому викладач має обрати один з двох варіантів: на лекції безпосередньо запропонувати студентам за інструкцією виконати дії зі створення першої програми. Другим варіантом може бути роздатковий матеріал, на якому інструкції зі скріншотами допоможуть студентам самостійно вдома виконати таке ж завдання.

Третя лекція продовжує далі ознайомлювати з інтерфейсом середовища та інструментами в ньому. Знову наводиться приклади програм, які створюються викладачем.

План лекції

I. **Організаційний момент**
II. **Оголошення теми, завдань курсу**
III. **Вивчення нового матеріалу**
IV. **Підведення підсумків лекції**

вид лекції: демонстраційна(пояснення з використанням проектору)

План лекції такий самий як і у минулій: лекція починається з невеликого опитування стосовно теми курсу та вже вивченого на минулих лекціях і далі йде вивчення середовища App Inventor на прикладі створення додатків. На цій лекції буде вивчено також два додатки: «Кольорові списки» (рис 2.7, 2.8) та «Розпізнання мови».

Рис. 2.7. Вигляд програми «Кольорові списки».

Рис. 2.8. Блоки програми «Кольорові списки».

Даний додаток має на меті показати, які типи списків існують в середовищі App Inventor, як їх створювати та налаштовувати.

Наступна програма «Розпізнання мови»(рис. 2.9, 2.10):

Рис. 2.9. Вигляд програми «Розпізнання мови».

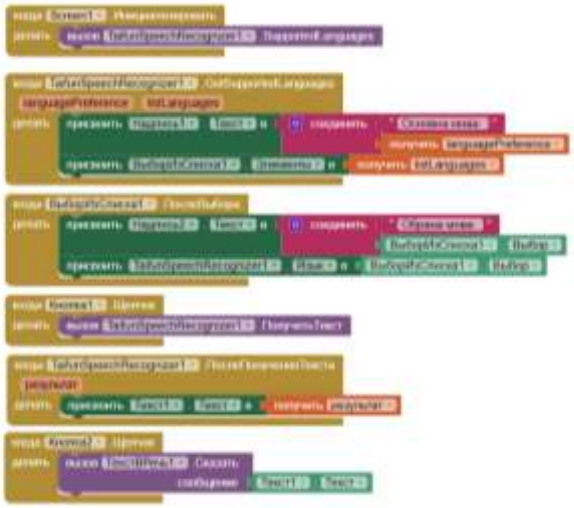

Рис. 2.10. Блоки програми «Розпізнання мови».

Дана програма показує, що в середовище можна інтегрувати сторонні розширення, які добавляють нові елементи з якими можна потім працювати. На цьому лекції закінчуються, починаються лабораторні роботи.

На лекціях студенти ознайомилися з інтерфейсом середовища, спостерігали за процесом створення додатків а також деякі із завдань повторили за викладачем. Можна сказати, вони вже мають основу для роботи в середовищі самостійно. Далі розпочинаються лабораторні роботи.

На першій лабораторній роботі студентам пропонується виконання одного завдання але двома способами. Додаток повинен обчислювати факторіал введеного користувачем числа(рис. 2.11). Перед початком роботи викладач повинен показати вигляд програми та її роботу. Блоки команд не демонструються.

Рис. 2.11. Вигляд додатку «Факторіал».

Оскільки завдання передбачає виконання двома способами, то і додатків повинно бути два(рис. 2.12, 2.13). Вигляд може бути однаковим проте блоки команд - різні. Для виконання завдання студент має використати цикли: з лічильником та з умовою.

Рис. 2.12. Блоки команд «Факторіал»(цикл з лічильником).

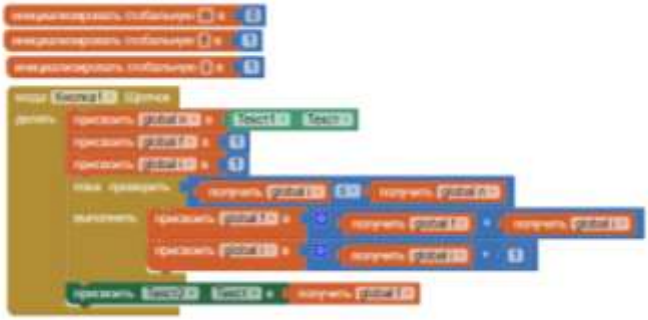

Рис. 2.13. Блоки команд «Факторіал»(цикл з умовою).

Як можна побачити блоки команд схожі, проте виконуються різні обчислення та використовуються різні блоки для кожного з циклів. Це завдання розвиває уміння працювати з циклами. Після виконання завдань студенти демонструють роботу додатку на емуляторі або смартфоні.

На наступній лабораторній роботі студентам пропонується виконати два завдання: «Таймер» та «Крокомір». Перша програма виводить повідомлення з текстом, який було введено користувачем, про завершення таймеру коли введений користувачем час вичерпано(рис. 2.14, 2.15). На повідомленні, для прикладу, наведено дві клавіши, при натисканні будь-якої з них повідомлення зникає а на головному екрані відображається яка кнопка була натиснута. Друга програма, як зрозуміло з назви, буде рахувати кроки користувача, а також

пройдену дистанцію та витрачені калорії. Оскільки в додатках з’являються нові елементи, то викладач повинен описати їх як це було з іншими елементами на лекціях.

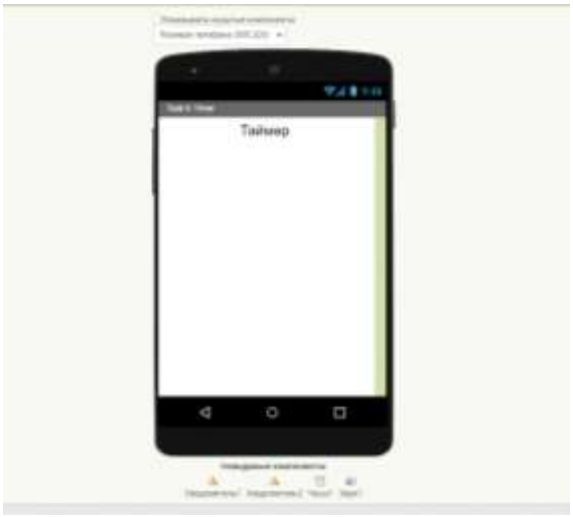

Рис. 2.14. Вигляд додатку «Таймер».

Програма містить такі елементи як: «Надпись», «Увидомитель», «Часы» та «Звук». З елементом «Надпись» студенти вже працювали, а «Часы», «Звук» та «Уведомитель» з'являються вперше.

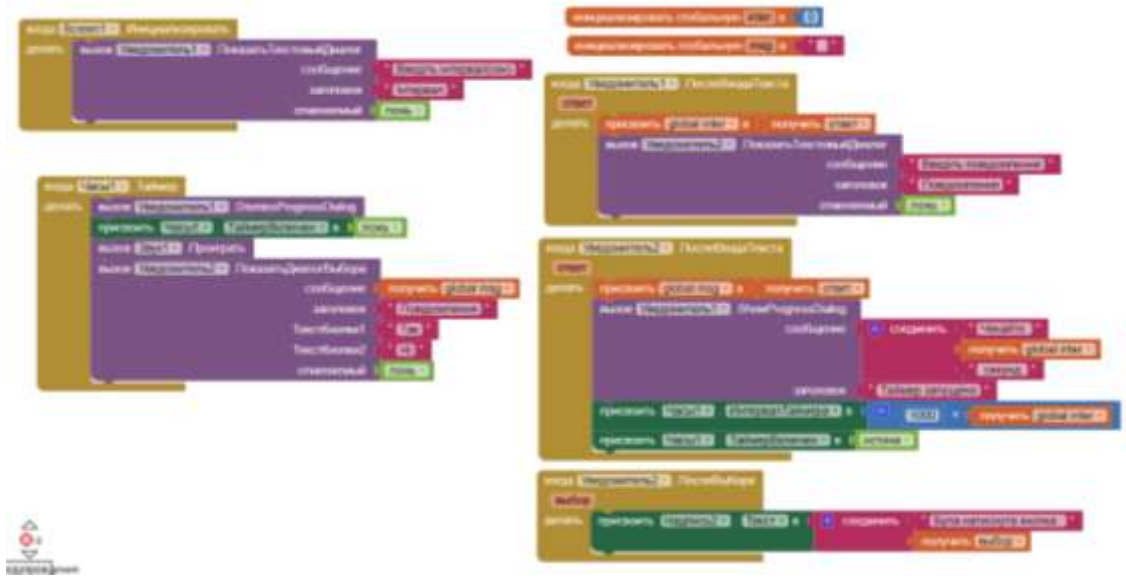

Рис. 2.15. Блоки додатку «Таймер».

Створення програми знайомить з новими елементами та вчить з ними працювати, розвивають вміння працювати в середовищі та алгоритмічне мислення.

Наступний додаток - «Крокомір»(рис 2.16, 2.17). Як вже було описано вище програма рахує кроки, пройдену дистанцію та витрачені калорії. Кроки рахуються новим елементом «Pedometr» а все інше обчислюється формулами з отриманих даних: зріст, вага та кількість кроків. Зріст та вага вводяться користувачем.

Рис. 2.16. Вигляд додатку «Крокомір».

Щоб продемонструвати роботу цього додатку потрібен смартфон, на емуляторі комп’ютера протестувати буде неможливо.

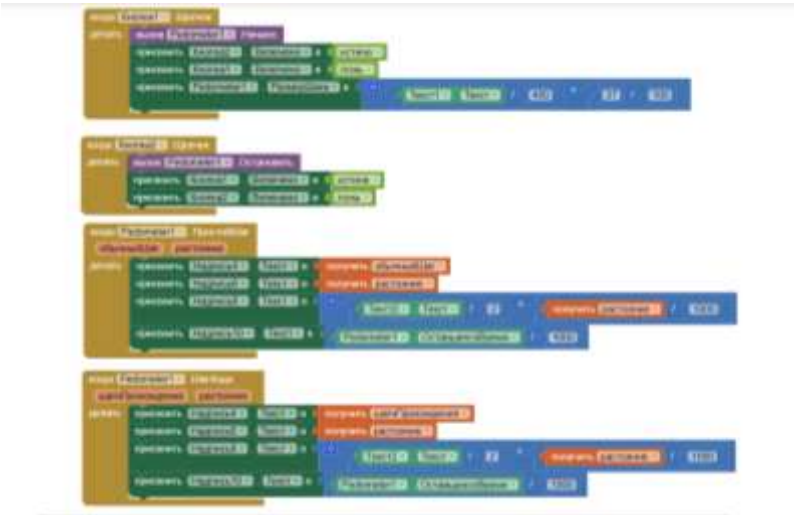

Рис. 2.17. Блоки додатку «Крокомір».

Створення цієї програми знайомить з новим елементом, закріплює вміння працювати вже з відомими елементами. Продовжується розвиток вміння працювати в середовищі.

На цьому завершується друга лабораторна робота. На ній студенти познайомилися з декількома новими елементами, навчилися з ними працювати та краще орієнтуватися в середовищі.

На третій лабораторній роботі також пропонується виконання двох завдань: «Словник» та «Paint 0,5». Робота відбувається аналогічно минулих лабораторних - викладач демонструє вигляд та роботу додатку, потім студенти працюють самостійно створюючи ці програми. Першою програма, що створюється - «Словник»(рис. 2.18).

Рис. 2.18. Вигляд програми «Словник».

Програма складається з двох екранів і містить елементи на обох. Це дасть уявлення студентам, як створювати додатки в яких може бети декілька екранів.

Оскільки програма містить два екрани, то до кожного екрану потрібно створювати блоки команд які будуть на ньому виконуватися(рис. 2.19, 2.20).

Рис. 2.19 Блоки команд першого екрану «Словник».

Рис. 2.20. Блоки команд другого екрану «Словник».

На першому екрані створюється список слів, які можна перекласти. А на другому екрані буде відображатися переклад слова, яке було обрано на першому екрані, також є кнопка повернення на перший екран. Виконуючи це завдання студенти навчаються працювати відразу з двома екранами.

Другою програмою буде - «Paint 0,5»(рис. 2.21, 2.22). Цей додаток дозволяє малювати на екрані. Користувач може обрати колір, яким малювати, а також є кнопка щоб зітерти полотно.

Рис. 2.21. Вигляд програми «Paint 0,5».

В цьому додатку студенти зустрічають новий елемент «Холст». Саме він є місцем для малювання та дозволяє це робити.

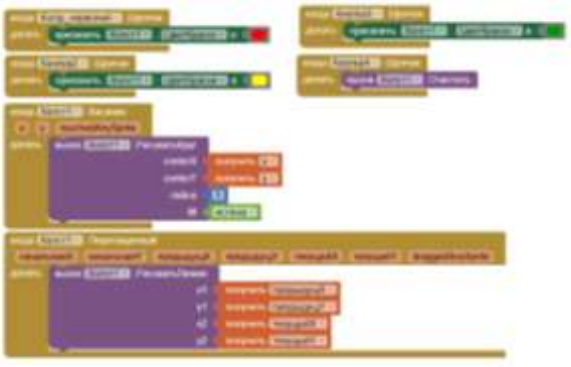

Рис. 2.22. Блоки команд «Paint 0,5».

Студенти навчаються працювати з новим елементом, його блоками. Продовжується розвиток умінь та навичок роботи у середовищі.

На цьому завершується третя лабораторна робота, студенти вивчили нові елементи покращили свої навички.

На четверту та п’яту лабораторну роботу пропонується виконання одного завдання. Воно буде складнішим та громіздким в порівнянні з минулим завданням, тому і на виконання дається два заняття. Додаток має назву «Відбивайка»(рис. 2.23, 2.24). Він представляє собою гру, в якій користувач за допомогою платформи відбиває шар і при кожному відбиванні йому нараховуються бали, які відображаються у верхній частині екрану. Відбивання повинно супроводжуватися звуком. Якщо користувач не відбиває шар і він дотикається нижньої частини екрану - гра завершується і користувач бачить повідомлення в якому вказано, що гру завершено.

Рис. 2.23. Вигляд додатку «Відбивайка»

З’являється новий елемент «Шар». Тому викладач повинен описати властивості цього елементу.

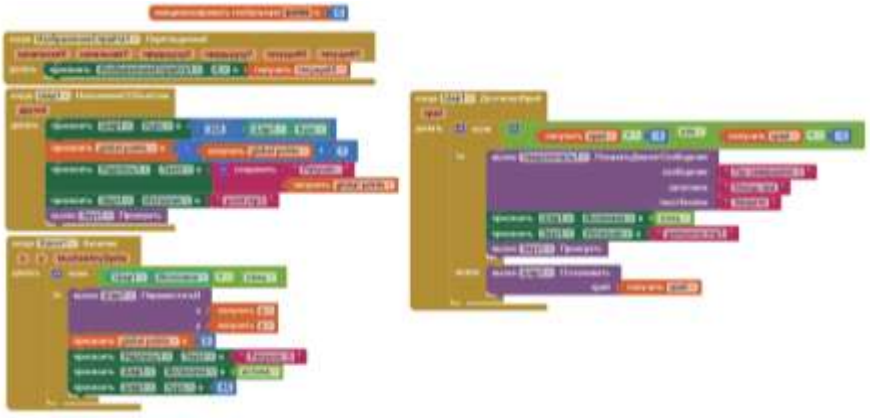

Рис. 2.24. Блоки команд гри «Відбивайка»

Як можна побачити блоки доволі громіздкі і складні, тому у студентів можуть виникати питання стосовно будови програми. Викладач має допомагати студентам, проте не потрібно відразу говорити які блоки потрібно використовувати та в якому порядку. Студент повинен намагатися зрозуміти це самостійно, на це і відводиться відразу два заняття. В цьому завданні користувач взаємодіє із зображенням платформи, він може його рухати дотиком до нього. Раніше студенти стикалися з тим, як добавляти зображення до програми, а тут програмується вже взаємодія з ним. Ну і звичайно новий елемент «Шар», з якими вони раніше не працювали, розширює знання в середовищі.

Виконуючи це завдання студенти знайомляться з новими елементи, дізнаються нове про вже відомі елементи. Розвивається вміння працювати в середовищі візуального програмування, закріплюються вже набуті знання.

На цьому завершуються завдання лабораторних робіт, далі дається завдання студентам - розробити власні додатки і продемонструвати їх роботу. Для виконання цього завдання відводиться три лабораторних роботи, звичайно студенти за бажанням можуть працювати і у свій вільний час.

2.2 Розробки студентів

Після створення власних додатків кожен студент має показати його та продемонструвати те, як вони працюють. Студент повинен окрім самого додатку показати і блоки команд. Для цього йому потрібно експортувати із середовища файл додатку(має розширення .aia) і здати його викладачу. Перевірка виконаних завдань може проходити, наприклад, у вигляді презентації. Студенти по черзі показують свої розробки аудиторії, демонструють їх роботу та відповідають на питання своїх одногрупників чи викладачів. Проводячи перевірку саме таким способом можна легко зрозуміти як добре той чи інший студент оволодів середовищем, як він в ньому орієнтується, та і взагалі, чи самостійно він створив свій додаток.

Нижче буде показано та описано деякі з додатків створених студентами 414 групи фізико-математично факультету Сумського державного педагогічного університету імені А.С. Макаренка.

Рис. 2.25. Вигляд додатку «Animal».

Першим розглянемо додаток «Animal»(рис. 2.25). На екрані додатку ми бачимо зображення тварин, з використання табличного розташування. Суть його полягає в тому, що при натисканні користувачем на зображення тварини, буде програвати звук цієї тварини. Наприклад, при натисканні на індика буде програвати звук індика, при натисканні на коня - звук коня, і так далі. На

скриншоті можна відразу спостерігати компоненти які були використані студентом в створенні і частину медіа файлів(зображення та звуки тварин). Це такі компоненти як: кнопка, табличне розташування а також «невидимий» елемент звук. За допомогою властивостей компоненту кнопка, було замінено її стандартний вигляд на зображення тварини. Також можна відмінити, що всі кнопки мають однакові розміри. Це говорить про те, що студент був уважним при розробці додатку.

Рис. 2.26. Блоки команд додатку «Animal».

Блоки команд можна побачити на наступному скриншоті(рис. 2.26). Було використано по однаковому блоку до кожної з кнопок. Всі дії, які виконують при натисканні на кнопку, також однакові. Йде відтворення відповідного звуку, який завантажив студент в додаток. Переглянувши екран блоків додатку, можна зробити висновок, що додаток доволі простий. В ньому немає розгалужень, виконання складних операцій. Лише виконується певна дія(програвання звуку) при натисканні на кнопку.

Наступний додаток має назву «Circle»(рис. 2.27). Екран цього додатку містить такі компоненти: «холст», вертикальне та горизонтальне розміщення, дві кнопки і таймер.

Рис. 2.27. Вигляд додатку «Circle».

Суть цього додатку полягає в наступному: при натисканні користувачем кнопки «Почати» програма запускається та відразу на місці для малювання починають з’являтися круги. Ці круги мають випадковий розмір, колір і можуть з’явитися в будь-якому місці, тому через деякий час з’являються поверх один одного. Круги виникають з певною періодичністю, про яку більш детально буде описано при розгляді екрану блоків. Коли процес запущено - кнопка «Почати» змінює назву на «Пауза» і, як можна здогадатися, при її натисканні нові круги більше не з’являються. Також коли кнопка паузи була натиснута, вона знову міняє назву, тепер вже на «Продовжити». Поряд завжди знаходиться кнопка «Очистити поле», яка стирає всі намальовані круги. Тепер переглянемо блоки команд цього додатку(рис. 2.28).

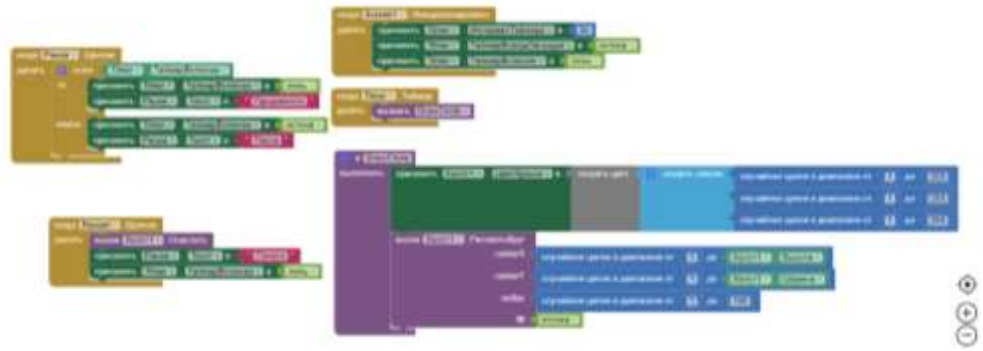

Рис. 2.28. Блоки команд «Circle».

Екран блоків виглядає не таким простим, як робота програми. Тут є блоки команд для кнопок, таймеру. А також використано процедуру для створення кругів. Якщо переглянути її команди то можна зрозуміти яким чином ці круги створюються. Що стосується блоку таймеру, то бачимо, що саме тут задається інтервал з яким створюються круги. Доволі цікаво реалізовано блоки кнопок. Насправді кнопка «Почати» - це «Пауза» і присвоєння їй цього надпису відбувається після натискання «Очистити поле». При запуску додатку відразу йде перевірка чи запущено таймер, через що програма не починає відразу створювати круги а лише після натискання кнопки. Коли таймер запущено, то починається виконання процедури створення кругів. В цьому додатку реалізовано багато різних команд, є перевірка умови, використовується процедура. Студент, який виконав це завдання, добре оволодів навичками роботи в цьому середовищі.

Далі розглянемо додаток «Draw1»(рис. 2.29). Відразу на екрані користувач бачить палітру з кольорами, що дає нам зрозуміти, що цей додаток дозволяє малювати.

Рис. 2.29. Вигляд додатку «Draw1».

У цьому додатку є такі елементи: «кнопка», «табличное расположение» та «горизонтальное расположение», «надпись», «холст» та «бегунок». Кожна з кнопок зафарбована у певний колір, натискаючи на нього користувач буде ним малювати. За допомогою компоненту «бегунок» можна змінити ширину лінії. Також можна зітерти малюнок і зберегти його на пристрій.

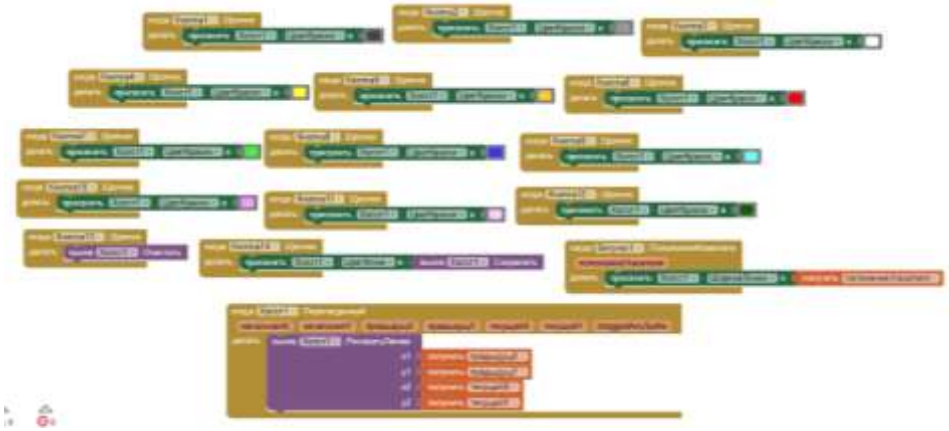

Рис. 2.30. Блоки команд «Draw1».

Екран розробки блоків можна побачити на наступному скріншоті(рис. 2.30). Тут можна побачити як відбувається робота додатку. Кнопки присвоюють колір лінії, і на блоках компоненту «холст» можна бачити як ця лінія визначається. Видно як за допомогою «бегунок» змінюється ширина лінії. Додаток не має складних зв'язків між компонентами. Виконується звичні операції для кнопок. Проте використовуються елемент «бегунок» і створене зображення можна зберегти на пристрій, що робить додаток більш цікавим.

Наступний додаток має назву «Indz»(рис. 2.31). Він представляє собою гру. Схожу програму студенти створювали на лабораторній роботі. Проте цей додаток містить декілька екранів і взагалі він більш складний.

Рис. 2.31. перший та третій екран «Indz».

На скріншоті вище можна побачити перший та третій екрани додатку. На першому знаходяться три кнопки. «Старт» відповідає за початок гри і

перемикає на другий екран додатку(рис. 2.32), на якому знаходиться сама гра. «Опции» перемикає на третій екран, тут користувач за допомогою компоненту «бегунок» може змінити гучність музики.

Рис. 2.32. Другий екран «Indz».

У цьому додатку використовується такі компоненти: «кнопка», «надпись», «изображение», «холст», «шар», «уведомитель», «бегунок», «проигрыватель» та «часы». Принцип гри такий самий як і в одному із завдань лабораторних робіт: м’яч рухається по екрану, відскакує від сторін екрану і при відбитті користувачем, за допомогою зображення платформи, нараховуються бали. Але тут є і перешкоди у вигляді блоків, коли м’яч відбивається від них теж нараховуються бали. Що стосоється блоків, оскільки екранів три, то до кожного екрану є відповідні блоки команд.(див. 2.33)

Рис. 2.33. Блоки першого(зліва) та третього(справа) екранів.

На цьому скріншоті видно блоки першого та третього екранів. На першому з елементів тільки кнопки, які відповідають за перехід між екранами

та кнопка виходу з додатку. Третій екран дозволяє змінити гучність музики і має відпвідні блоки. На другому екрані найбільша кількість блоків, адже тут сама гра(див. 2.34).

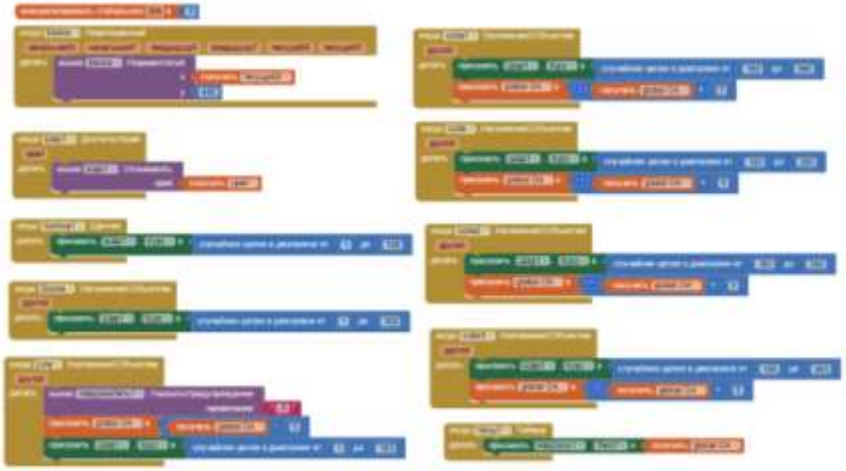

Рис. 2.34. Блоки другого екрану.

Тут можна побачити як запрограмовано рух м'яча, нарахування та списання балів. Щойно користувач переходить на цей екран м'яч, за вмовчування, рухається від лівого краю до правого. Коли натиснута кнопка «Старт», м’яч починає рух вгору і можна грати. Цей додаток вже більш складний, використовується багато компонентів та декілька екранів. Можна зрозуміти, що студент, який створив цей додаток, оволодів цим середовищем досить добре.

Далі розглянемо додаток «INDZ21»(рис. 2.35). Цей додаток працює як перекладач. Він дозволяє робити переклад з російської мови на англійську та навпаки.

Рис. 2.35. Вигляд додатку «INDZ21».

Цей додаток містить такі компоненти: «кнопка», «надпись», «распознаватель речи», «текст в речь», «Яндекс переводчик» та «switch». Для того, щоб почався переклад на англійську мову користувачу потрібно натиснути на кнопку «Переклад на англійську мову(English translation)» і сказати щось на російській мові. Додаток розпізнає слова, відображає на екрані, перекладає на англійську і озвучує переклад. Аналогічно працює і переклад з англійської на російську - користувач натискає на кнопку «Russian translation(Переклад на російську мову)», озвучує англійський текст і чує переклад російською мовою. Також є перемикач, який збільшує розмір шрифту. Блоки команд можна бачити на наступному скріншоті(рис. 2.36).

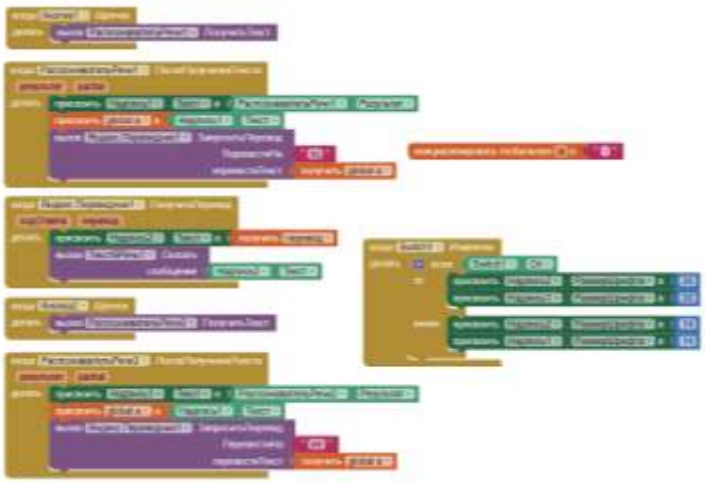

Рис. 2.36. Блоки команд «INDZ21».

Бачимо як працює додаток, кнопки запускають відповідне розпізнання мови, це присвоюється в надписи і озвучується. В цьому додатку був використаний «switch» - це звичайний перемикач, в даному випадку він змінює розмір шрифту, реалізований за допомогою перевірки умови.

Ще один з додатків «Spiner»(рис. 2.37). Цей додаток дозволяє користувачу погратися досить відомою іграшкою - спінером.

Рис. 2.37. Вигляд додатку «Spiner».

Цей додаток містить такі компоненти: «горизонтальное расположение», «бегунок», «надпись», «холст», «часы», «проигрыватель» та «изображение». Користувач може регулювати швидкість руху та силу тертя спінера, від чого залежить те, як він буде обертатися. Для того, щоб спінер почав рух достатньо провести по ньому в будь-яку сторону, чим частіші та швидші рухи - тим вища і швидкість оберту. Внизу є зображення лампи, вона загорається коли спінер починає рух, і разом з цим починає відтворюватися музика. Як тільки спінер зупиняється, лампа вимикається і музика затихає. Додаток має доволі складну будову і тому в ньому багато блоків(рис. 2.38.).

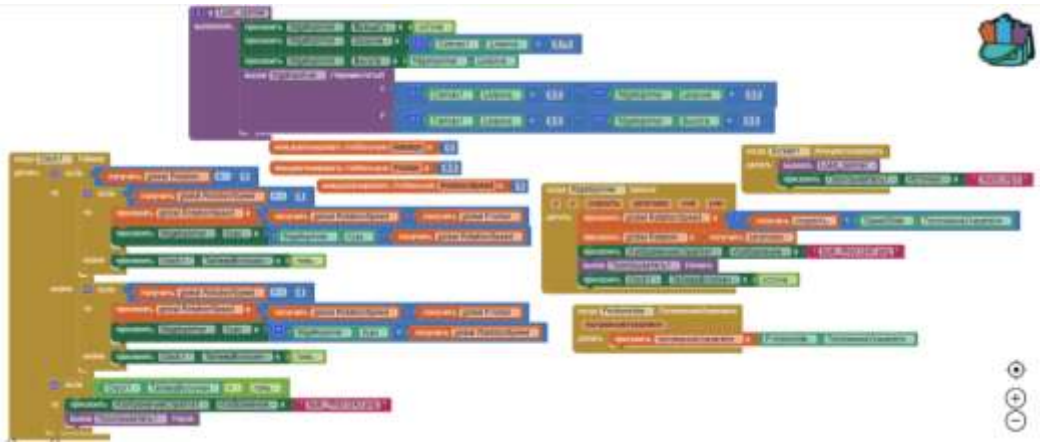

Рис. 2.38. Блоки команд «Spiner».

У блоках команд можна побачити як рахується швидкість, сила тертя спінера. Перевіряється умова чи взагалі він обертається для того, щоб програма розуміла чи вмикати лампу та увімкнути музику. Додаток дійсно складний і має багато зв’язаних компонентів, їх зв’язок тут також видно.

Наступний додаток має назву «trigonometria2»(рис. 2.39). Додаток представляє собою калькулятор тригонометричних функцій, таких як sin, cos, tg(синус, косинус, тангенс).

Рис. 2.39. Вигляд додатку «trigonometria2».

У цьому додатку використано такі компоненти: «надпись», «текст», «кнопка», «вертикальное расположение». Програма доволі просто працює - користувач вводить значення яке необхідно обчислити (x) та обирає, яку саме функцію потрібно знайти. Програма виводить відповідь на відповідне поле. На скріншоті цього не видно, проте у властивостях текстового поля є параметр

«подсказка», в ньому вказано яке поле відповідає за (x), а яке за відповідь. Також додаток містить кнопку «Очистити», яка дозволяє швидко очистити текстові поля. Блоки команд, як і сам додаток, виглядають просто(рис. 2.40).

Рис. 2.40. Блоки команд додатку «trigonometria2».

Все що міститься в блоках - це дії для натискання на кнопки. При натисканні на кнопку функції береться значення введене користувачем з відповідного текстового поля, рахується його значення та присвоюється у текстове поле відповіді. Так з кожною функцією. А при натисканні на «Очистити» текстовим полям присвоюється порожнє значення. Студент, який створив цей додаток, засвоїв роботу з найпоширенішими компонентами у середовищі.

Останній додаток який ми переглянемо - «Chatbot»(рис. 2.41). Ця програма дозволяє поспілкуватися з ботом.

Рис. 2.41. Вигляд додатку «Chatbot».

Даний додаток містить такі компоненти: «надпись», «кнопка», «табличное расположение», «горизонтальное расположение» і «текст». Для

того розпочати бесіду користувачу потрібно написати текст у відповідне поле та натиснути кнопку «Submit». Але бот має обмежену кількість відповідей, тому і користувач має обмежену кількість фраз, які мають сенс. Бот вміє: вітатися та питати як справи, називати своє ім’я, а коли він бачить невідому фразу, то відповідає, що не розуміє користувача. Блоки команд мають наступний вигляд(рис. 2.42).

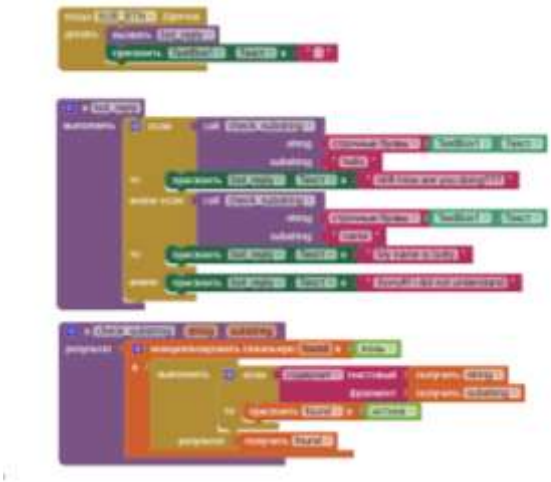

Рис. 2.42. Блоки команд додатку «Chatbot».

Додаток використовує дві процедури для роботи: перша процедура перевіряє весь текст введений користувачем; друга - шукає у введеному тексті слово на яке є відповідь і виводить відповідь бота на екран. Все це запускається при натисканні на кнопку «Submit». Доволі цікава структура додатку з порівнянням та розгалуженням. Було б більш цікавіше, якщо студент розробив більше відповідей для бота.

ВИСНОВКИ

Аналіз матеріалів з теми візуальне програмування надає такі результати:

- Візуальне програмування дозволяє створювати додатки та програми для різних операційних систем не за допомогою написання коду - а взаємодією з візуальними компонентами.
- Візуальне програмування варто пропонувати до вивчення текстових мов програмування, створення лістингу програм студентами;
- Вивчення та робота у візуальних середовищах програмування розвиває алгоритмічний стиль мислення. Даний тип мислення дозволяє розв’язувати задачі, що виникають у будь-якій сфері діяльності людини, не тільки в програмуванні. Добре розвинений алгоритмічний стиль мислення, вміння мислити точно, стає одними з важливих ознак загальної культури людини в цьому цифровому світі.
- Розвиток навичок у візуальному програмуванні буде розвивати навички взагалі у програмуванні. Тому навчання візуальному програмуванню є досить корисним.

Ознайомившись та вивчивши середовище візуального програмування MIT App inventor 2 можна зрозуміти наступне:

- MIT App Inventor 2 це середовище візуального програмування для розробки додатків для операційної системи Android.
- Для роботи в середовищі MIT App Inventor 2 не обов’язково володіти текстовими мовами програмування.
- Середовище App Inventor 2 дозволяє, за допомогою супутнього додатку MIT App Inventor 2 Companion, швидко тестувати та редагувати створені додатки в режимі реального часу.
- Робота в цьому середовищі буде корисна як студентам, які далі хочуть розвиватися в програмуванні, так і тим студентам, хто не має бажання працювати в цій сфері.

Стосовно викладання візуального програмування у ЗВО, то проаналізувавши наукові та методичні матеріали з теми можна зробити висновки:

- Вивчення візуального програмування на даний момент не поширене.
- Теоретичний матеріал, що викладається, потрібно закріплювати практично, тому викладач має обрати один з двох варіантів: на лекції безпосередньо запропонувати студентам за інструкцією виконати дії зі створення вивченої програми. Другим варіантом може бути роздатковий матеріал, на якому інструкції зі скріншотами допоможуть студентам самостійно вдома виконати таке ж завдання.
- При розробці курсу для студентів ЗВО потрібно оволодіти середовищем в якому вони будуть працювати, пояснити важливість та актуальність вивчення даної теми.
- Були розроблені та впроваджені в освітній процес матеріали до лекційних та лабораторних занять.

Таким чином за допомогою середовища MIT App Inventor можна ознайомити та навчити студентів візуальному програмуванню, що стане у нагоді в майбутньому вивченні програмування. В цілому поставлена мета досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Visual programming URL: https://www.digitaltechnologieshub.edu.au/teachers/topics/visual-programming
2. Візуальне програмування URL: http://www.wikiwand.com/uk/Візуальне_програмування
3. Visual programming language URL: https://en.wikipedia.org/wiki/Visual_programming_language#External_links
4. What is visual programming? URL: https://bitspark.de/blog/what-is-visual-programming
5. What Is Visual Programming? URL: https://www.outsystems.com/blog/posts/what-is-visual-programming/
6. A History of Visual Programming URL: https://bubble.io/blog/visual-programming/
7. Ковальчук М.Б. Змістові аспекти алгоритмічного мислення. *Фізико-математична освіта*. 2018. Випуск 3(17). С. 61-66.
8. Алгоритмічний стиль мислення URL: https://studfile.net/preview/7517256/page:6/
9. Repenning, Alexander "Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets". *Journal of Visual Languages and Sentient Systems*. 2017. С. 68-91 - URL: http://ksiresearchorg.ipage.com/vlss/journal/VLSS2017/vlss17paper_10.pdf
10. The maturity of visual programming URL: https://www.craft.ai/blog/the-maturity-of-visual-programming
11. New clues emerge about Amazon’s secretive low-code/no-code project URL: https://www.geekwire.com/2019/aws-everyone-new-clues-emerge-amazons-secretive-low-code-no-code-project/
12. App Inventor URL: https://uk.wikipedia.org/wiki/App_Inventor
13. App Inventor for Android URL: https://en.wikipedia.org/wiki/App_Inventor_for_Android

14. MIT App Inventor: Objectives, Design, and Development URL: https://link.springer.com/chapter/10.1007/978-981-13-6528-7_3

15. To block or not to block, that is the question: students' perceptions of blocks-based programming URL: https://dl.acm.org/doi/abs/10.1145/2771839.2771860

16. App Inventor Java Bridge URL: http://www.appinventor.org/jbridge

17. New frameworks for studying and assessing the development of computational thinking URL: http://scratched.gse.harvard.edu/ct/files/AERA2012.pdf