Рисунок 73. Друга чорна скринька

Використаємо такі назви:

- ***Початкова позиція***. Це початкове положення ***Vector2D*** об'єкта для запуску. Тут не обов'язково запитувати ці дані, але це дуже зручно, якщо ми хочемо обчислити остаточну позицію суб'єкта за певний час, а не тільки зсув. Таким чином, ми заощаджуємо за допомогою іншого чорного ящика, щоб зробити суму.
- ***швидкість***. Швидкість ***Vector2D*** елемента (розрахована і доступна у вихідному параметрі ***швидкості*** попереднього ***CodeRunner***).
- ***час***. Час, що минув з моменту запуску.
- ***позиція***. Вихідний параметр, де ми будемо зберігати розраховану позицію ***Vector2D*** у заданий час.

Вставимо наступний код:

```
var initialPosition  = inParams.initialPos,
vx = inParams.velocity.x,
vy = inParams.velocity.y,
time = inParams.time / 100;
var GRAVITY  = 9.8;
outParams.position.x = initialPosition.x  + vx * time;
outParams.position.y = initialPosition.y  + vy * time + GRAVITY/2  * time * time;
outEvents.done();
```

Отримаємо щось подібне (рис. 73).

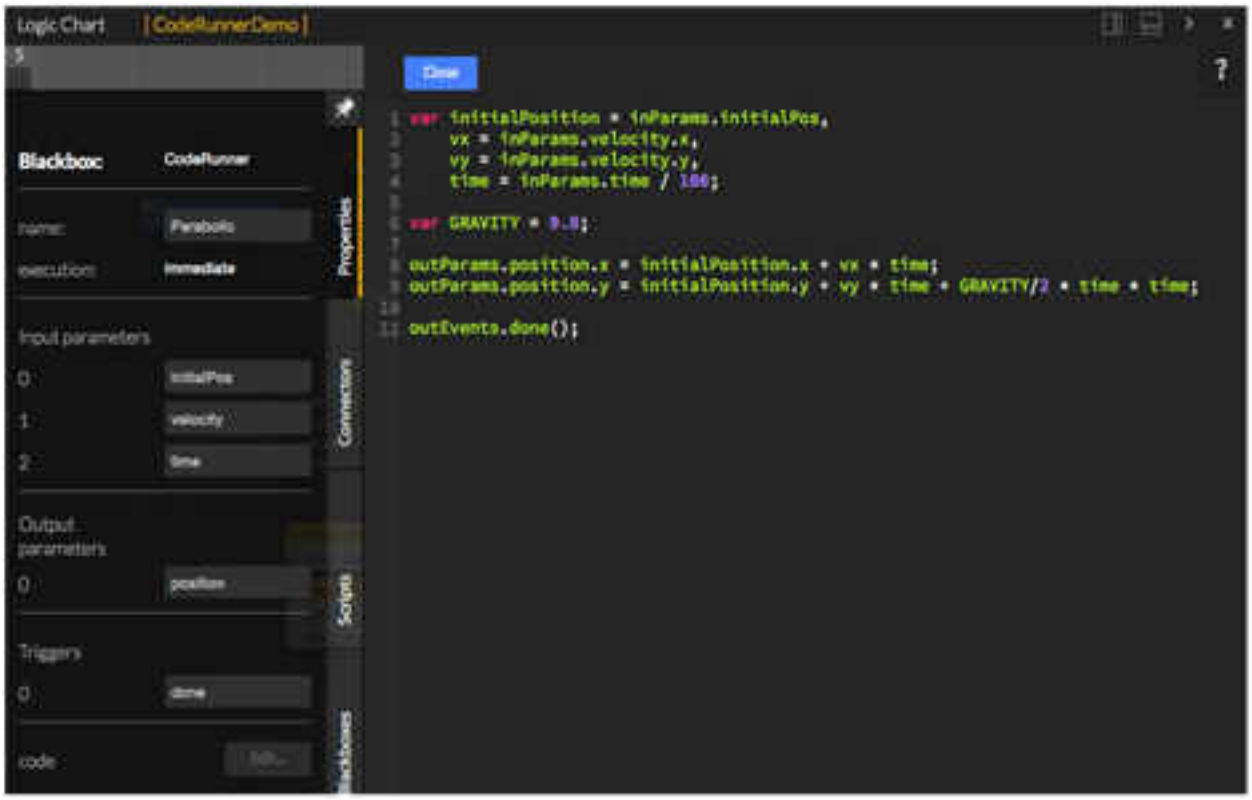

Рисунок 73. Результат запуску коду

Гаразд, ми маємо обидва ***CodeRunners*** готові до використання.

Потрібно об'єднати їх лише з іще трьома чорними скриньками, створюючи наступну діаграму (рис. 74).

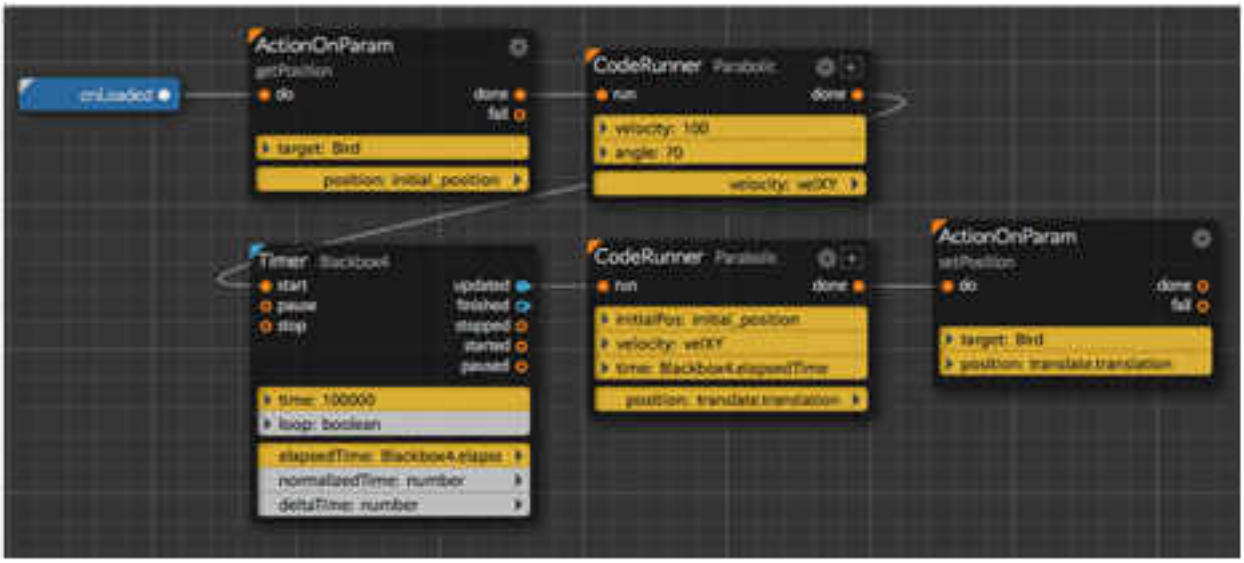

Рисунок 74. Діаграма чорних скриньок

Ми плануємо створити цю схему крок за кроком.

1. Нам потрібна сцена принаймні з одним об'єктом. Фактично, це об'єкт, який ми збираємося запустити. Отже, перейдіть до ***редактора сцени***, створіть сцену з об'єктом і поверніться до ***редактора логіки***.
2. Перетягніть об'єкт, створений на попередньому кроці, у ***логічну діаграму***. У нашому випадку його прізвище ***ntax***. Новий ***ActionOnParam*** blackbox буде створено автоматично за допомогою ***клону*** дії, вибраної за замовчуванням. Перейдіть до його властивостей та виберіть дію ***getPosition*** у списку властивостей ***дії***. За допомогою цієї дії ми отримуємо позицію об'єкта. Ми використаємо ці дані пізніше. Підключіть connector ***onLoaded*** до активатора ***do***. Потім підключіть ***виконаний*** тригер до активатора ***запуску*** в першому ***CodeRunner***.
3. Перетягніть чорну скриньку ***таймера*** з розділу "***Час***" на вкладці "***Чорні ящики"***. Двічі клацніть на параметрі введення ***часу*** та змініть його значення на щось більше 10000. Це час, який ми будемо використовувати для виклику нашого CodeRunners. Коли закінчиться цей час, анімація зупиниться. Зауважте, що час вимірюється в мілісекундах. Тепер перетягніть вихідний параметр, що ***минув, Timer*** blackbox на параметр введення ***часу*** нашого другого ***CodeRunner***. Ви побачите, що дві нові змінні ***номера*** типу створюються автоматично.
4. Ми повинні призначити параметри другого ***CodeRunner***. Почнемо з ***initialPos***. У другому пункті ми створили ***BlackBox ActionOnParam*** з ***вибраною*** дією ***getPosition***. Перетягніть ***стан*** вихідного параметра і помістіть його на ***initialPos*** вхідного параметра другого ***CodeRunner***. Другий параметр – це x і y швидкість. Ми маємо цю інформацію в параметрі виведення ***швидкості*** першого створеного нами CodeRunner, тому перетягніть його звідти і викиньте його на параметр введення ***швидкості*** цього чорнила. Останним вхідним параметром є ***час***.
5. Ми майже закінчили. Потрібно лише призначити позицію, обчислену на об'єкт, який ми рухаємо. Отже, перетягніть об'єкт з ***диспетчера активів,*** щоб створити нову ***панель ActionOnParam***, але тепер вибирайте дію ***setPosition*** зі своїх властивостей. Нарешті, перетягніть параметр виведення ***позиції*** з другого CodeRunner на параметр введення ***позиції***.

   І це все.

Тепер натисніть кнопку відтворення на панелі попереднього перегляду, щоб побачити результат.

Ви повинні побачити елемент, який ви обрали після параболічної траєкторії, залежно від кута і швидкості, наданих у першому CodeRunner.

ЛЕКЦІЯ 4

ТЕСТУВАННЯ, РОЗГОРТАННЯ ТА ПУБЛІКАЦІЯ КОМП'ЮТЕРНОЇ ГРИ

ВІРТУАЛЬНІ ТОВАРИ

У цій лекції ми спочатку розглянемо, як правильно керувати віртуальними товарами в WiMi5 (Virtual Goods).

Virtual Goods – це елементи ігор, які дозволяють розробникам ігор HTML5 монетизувати свої відеоігри.

У бізнес-моделі Free to Play, Virtual Goods зазвичай є елементами, такими як віртуальні монети, дорогоцінні камені, дорогоцінні камені, бонуси або будь-який об'єкт, який може налаштувати ігровий персонаж або аватар.

Є багато можливостей для створення віртуальних товарів та монетизації відеоігор, єдиним обмеженням є креативність дизайнера гри.

ОГЛЯД ВІРТУАЛЬНОГО УПРАВЛІННЯ ТОВАРАМИ.

У WiMi5 Virtual Goods управляються з декількох місць.

Віртуальні товари створюються та налаштовуються на ***інформаційній панелі,*** після чого вони інтегруються в гру за допомогою чорних ящиків в ***редакторі логіки***.

У ***редакторі логіки*** також можна використовувати панель ***віртуальних товарів*** для налаштування параметрів під час розробки гри.

Крім того, можна переглянути і протестувати продуктивність віртуальних товарів з допомогою ***попереднього перегляду*** панелі в ***редакторі логіки***. Він покаже віртуальний кошик для покупок.

Давайте проаналізуємо ці процеси.

СТВОРЕННЯ ТА КОНФІГУРАЦІЯ ВІРТУАЛЬНИХ ТОВАРІВ

Віртуальні товари створюються на ***інформаційній панелі*** (рис. 75) у розділі "***Віртуальні товари***".

Коли ви вперше ввійдете в цей розділ, ви побачите таке вікно.

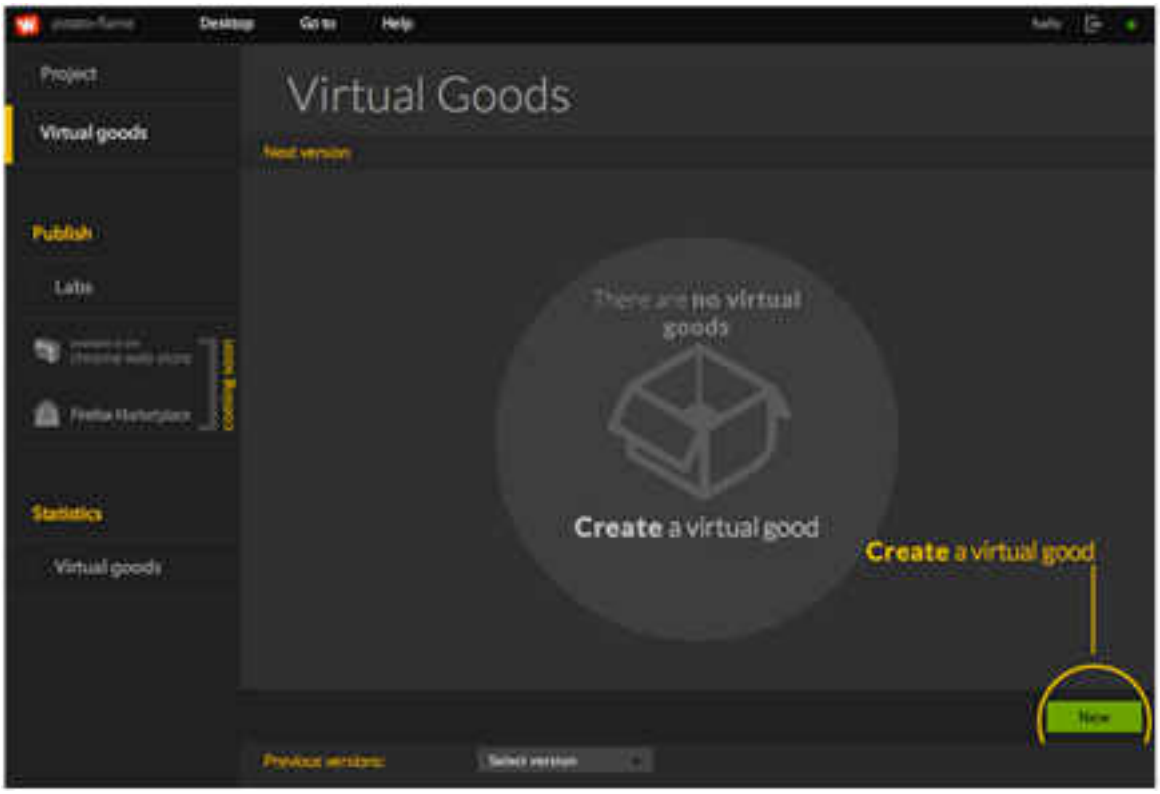

Рисунок 75. Панель Віртуальних товарів

Коли ви натискаєте кнопку "***Новий"***, створюється новий віртуальний товар із ціною за замовчуванням (0,99 €) та переглядом кошика (так) (рис. 76).

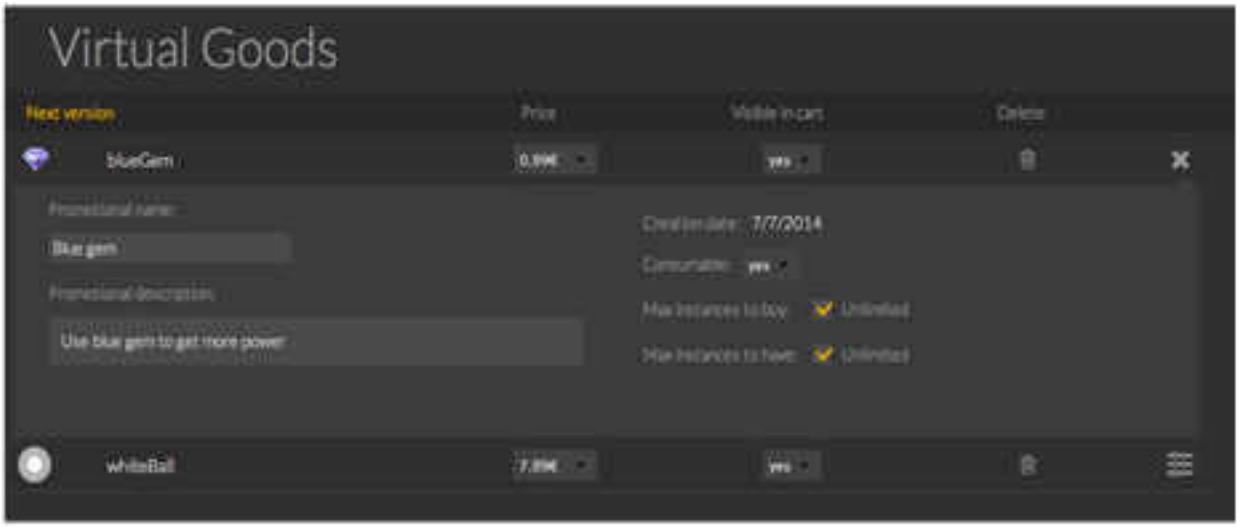

Рисунок 76. Створення віртуального товару

Значок кожного віртуального товару можна налаштувати, натиснувши логотип WiMi5, розташований ліворуч від значка товару.

Натиснувши кнопку ***Додаткові параметри,*** ви також можете переглянути та налаштувати додаткову інформацію: ***рекламну назву*** (ім'я

товару, яке буде показано гравцям), ***рекламний опис***, ***дату створення***, ***споживну*** поведінку та встановити максимальну кількість елементів, які можуть бути придбані або належати гравцю.

Наприклад, у грі ви можете запропонувати золоті монети. Якщо гравець може споживати монети, що зменшує їх кількість, ми кажемо, що цей віртуальний продукт є витратним.

З іншого боку, якщо ви пропонуєте надмірно потужний меч, який будь-який гравець може придбати та використати, але його обсяг не зменшується, ми кажемо, що це не є витратним товаром. Як тільки гравець отримає невитратний віртуальний продукт, він не може зникнути, гравець буде мати цей меч назавжди.

Ви також можете вибрати, використовуючи комбо в нижній частині вікна, ***попередні версії*** проекту та віртуальні товари, створені в кожній версії (рис. 77).

Після розгортання проекту неможливо видалити попередні віртуальні товари, оскільки гравець міг придбати віртуальний товар і досі не використав або не витратив його.

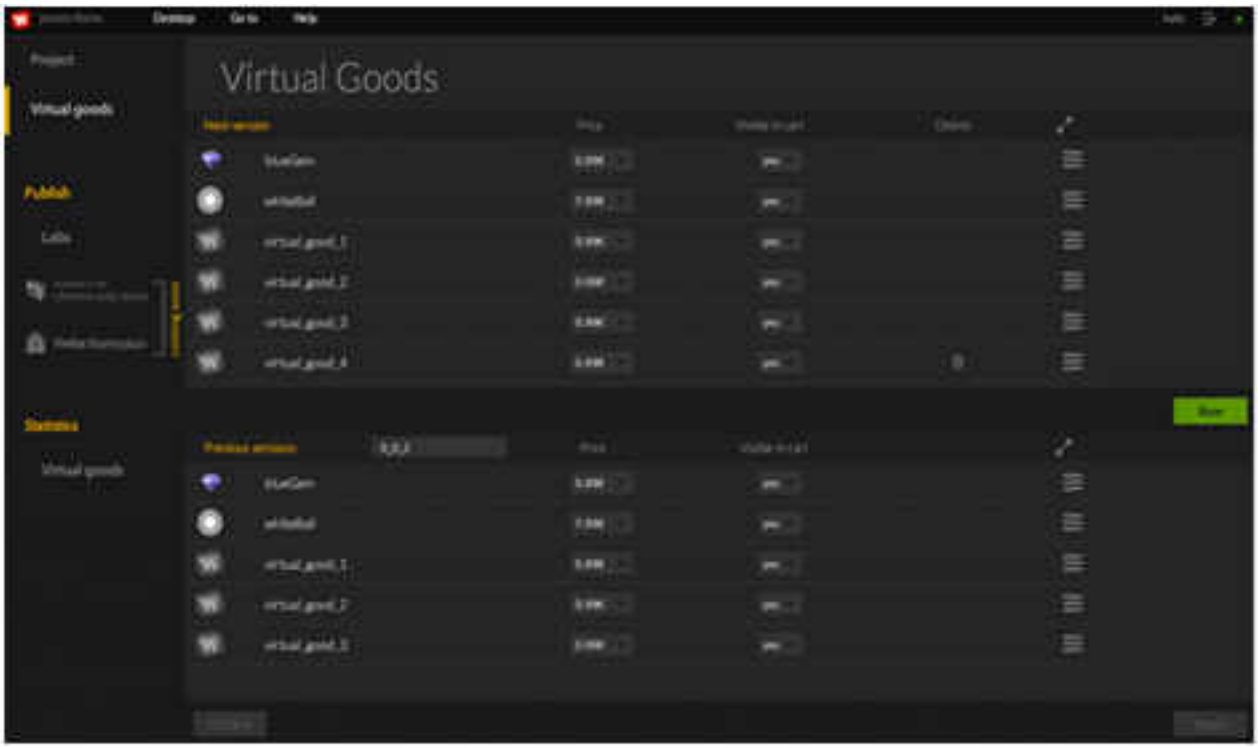

Рисунок 77. Використання віртуальних товарів

ІНТЕГРАЦІЯ ВІРТУАЛЬНИХ ТОВАРІВ В ЛОГІКУ ГРИ

Коли віртуальні товари створюються та правильно налаштовуються, розробники гри можуть визначити, як і коли віртуальні товари будуть використовуватися, купуватися, відображатись або витрачати.

Для цього розробники гри можуть використовувати набір чорних скриньок, які дозволяють керувати віртуальними товарами та інтегрувати їх у логіку гри.

Існує також новий тип ***параметру,*** який називається ***VirtualGood,*** і який може використовуватися в поєднанні з вищезгаданими чорними скриньками.

Це все можна зробити в ***редакторі логіки***, де ви можете знайти новий набір чорних скриньок у розділі «***Віртуальні товари»*** на вкладці «Чорні скриньки» (рис. 78).

Рисунок 78. Розділ Віртуальні товари у вкладці Чорні скриньки

ВІРТУАЛЬНІ ЧОРНІ СКРИНЬКИ

WiMi5 має набір з 3 чорних скриньок, які дозволяють виконати наступні дії:

- ***ConsumeVirtualGood***: за допомогою цієї ***чорної скриньки*** розробники можуть налаштувати дію споживання віртуального блага.
- ***GetVirtualGoodAmount***: за допомогою цього Blackbox можна встановити або отримати кількість одиниць, які будь-який віртуальний продукт має в певний момент.
- ***ShowShoppingCart***: ця ***чорна скринька*** дозволяє розробникам показувати кошик для покупок, що містить всі віртуальні товари, створені та налаштовані в налаштуваннях гри на ***інформаційній панелі***.

Кошик (рис. 79) буде показано на панелі ***попереднього перегляду*** як симуляція справжньої кошика, яка буде показана у ігровому середовищі.

Перш за все треба переконатися, що панель "***Віртуальні товари***" є видимою.

Ця панель може бути показана за допомогою елементів керування візуалізацією, розташованих у верхньому правому куті кожної панелі.

Якщо це не відображається, розділіть панель, натиснувши кнопку ***Додати над*** кнопкою ***панелі***, а потім у новій панелі виберіть «***Віртуальні товари»*** у кнопці «***Вибрати***».

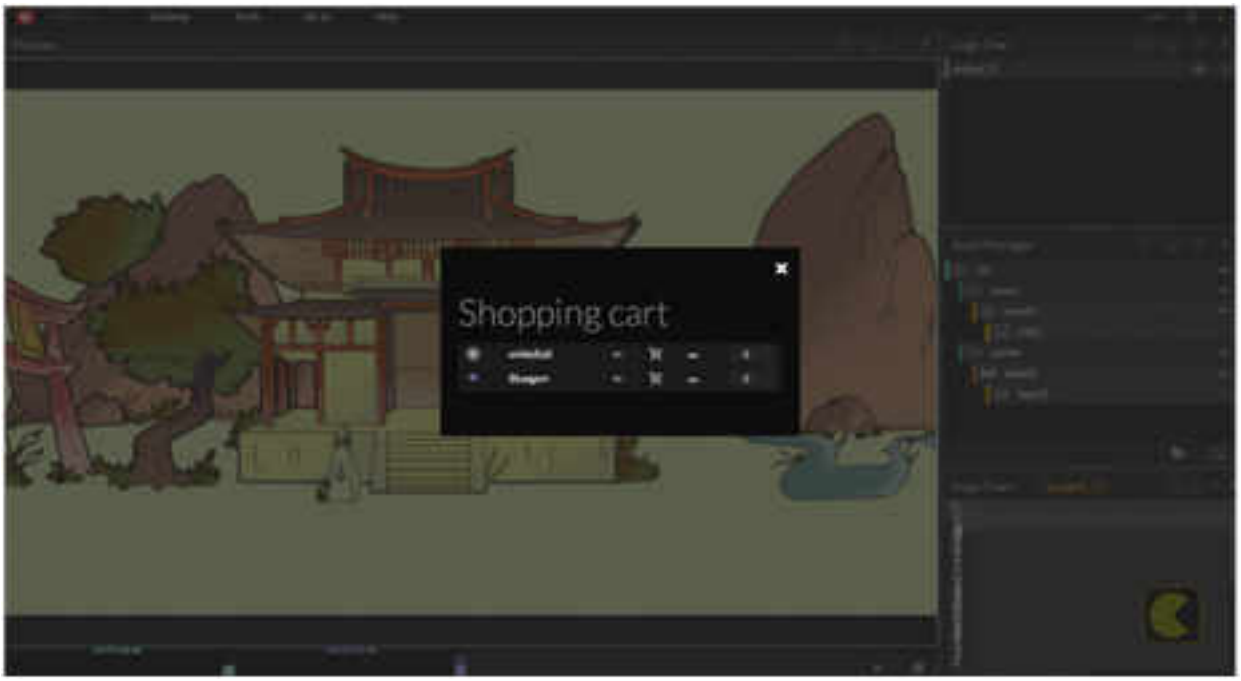

Рисунок 79. Кошик

Для використання цих чорних скриньок ви можете перетягувати параметри віртуальних товарів з панелі «***Віртуальні товари»***, або ви також можете перетягувати параметри типу віртуальних товарів з вкладки «***Параметри***», розташованої в лівій частині вікна ***редактора логіки***.

У ***віртуальних*** магазинах (рис. 80) можна також налаштувати кількість одиниць будь-якого віртуального товару. Під час виконання або попереднього перегляду будь-якої гри в редакторі ви можете побачити збільшення або зменшення всіх одиниць віртуального товару в цій панелі.

Таким чином, ви можете перевірити належне функціонування логіки гри, яка керує віртуальними товарами.

Рисунок 80. Віртуальний магазин

ПУБЛІКАЦІЯ ПРОЕКТІВ

Коли ви ввійдете в WiMi5, ви отримаєте доступ до **інформаційної панелі** (рис. 81), де ви зможете керувати своїми проектами.

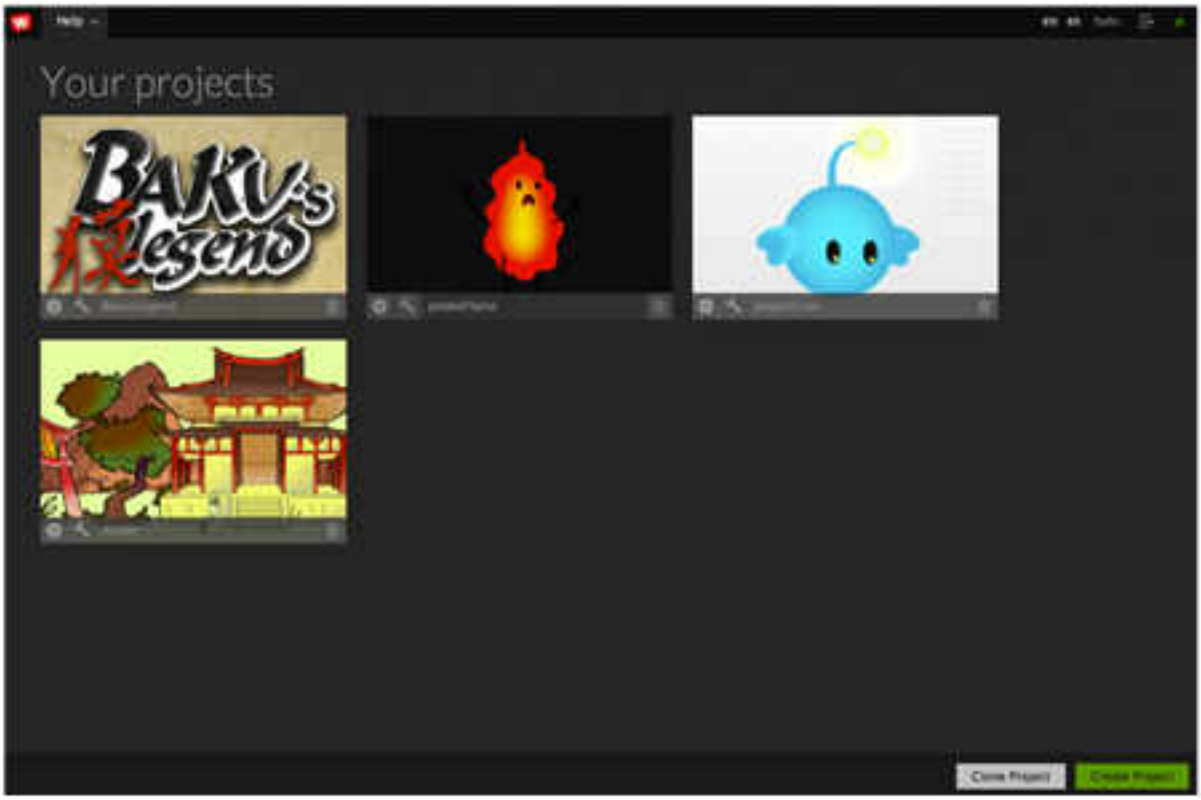

Рисунок 81. Інформаційна панель проекту

В даний час ви можете створити новий проект, редагувати існуючі або клонувати проект із сховища зразків проекту.

На інформаційній панелі ви знайдете усі створені проекти, і в кожному проекті є 3 кнопки (рис. 82).

Рисунок 82. Кнопки управління проектом

За допомогою лівої кнопки ви можете отримати доступ до налаштувань вашого проекту.

Правою кнопкою ви можете отримати доступ до інструменту створення ігор WiMi5 і почати редагувати свою гру HTML5.

У налаштуваннях ви зможете опублікувати свою гру, і в інструменті створення гри ви можете протестувати свою гру.

ТЕСТУВАННЯ HTML5-ІГРИ З WIMI5

Розробникам гри настійно рекомендується періодично перевіряти свою гру.

Ви можете скористатися панеллю попереднього перегляду, щоб швидко перевірити свою гру. Крім цього, WiMi5 також дозволяє дуже просто перевірити свою HTML5-версію.

Як пояснили раніше, ви завжди повинні розгортати свій проект, перш ніж публікувати його, щоб побачити нові зміни в грі.

Для розгортання проекту ви можете скористатись опцією "Розгортати" в меню "Сервіс" верхньої панелі (рис. 83).

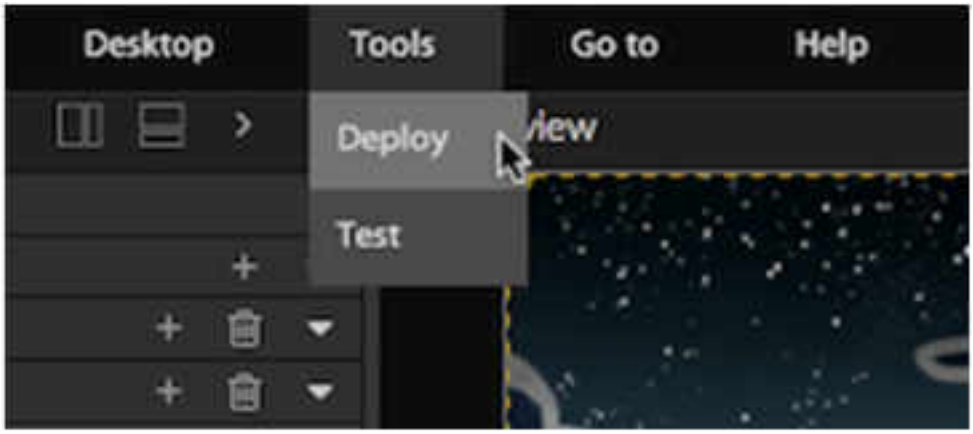

Рисунок 83. Панель Розгортання гри

Після розгортання ви отримаєте таке вікно (рис. 84).

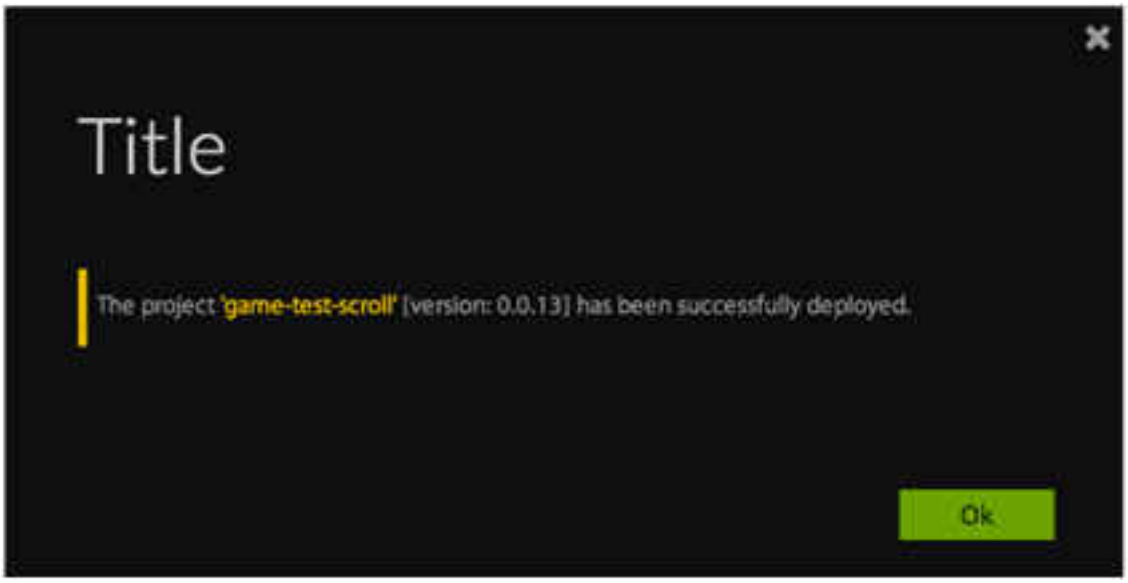

Рисунок 84. Результат виконання розгортання

Після розгортання проекту ви можете опублікувати свій проект у приватній URL-адресі за допомогою жовтої кнопки.

Платформа створить URL-адресу (рис. 85), яку ви можете використовувати для тестування або спільного використання з вашою командою або клієнтами.

Рисунок 85. URL-адреса проекту

ПУБЛІКАЦІЯ HTML5-ІГРИ З WIMI5

Для налагодження проекту в WiMi5 є 2 варіанти:

1. Використовуйте чорну скриньку CodeRunner, а всередині неї, напишіть код javascript для друку журналу на консолі переглядача (рис. 85).

Наприклад:

console.log ("este es mi log");

До цього коду можна додати значення параметрів, взяті з BlackBox CodeRunner. Тому, якщо у нас є вхідний параметр під назвою 'vidas', ми можемо написати:

console.log ("Número de vidas:" + inParams.vidas);

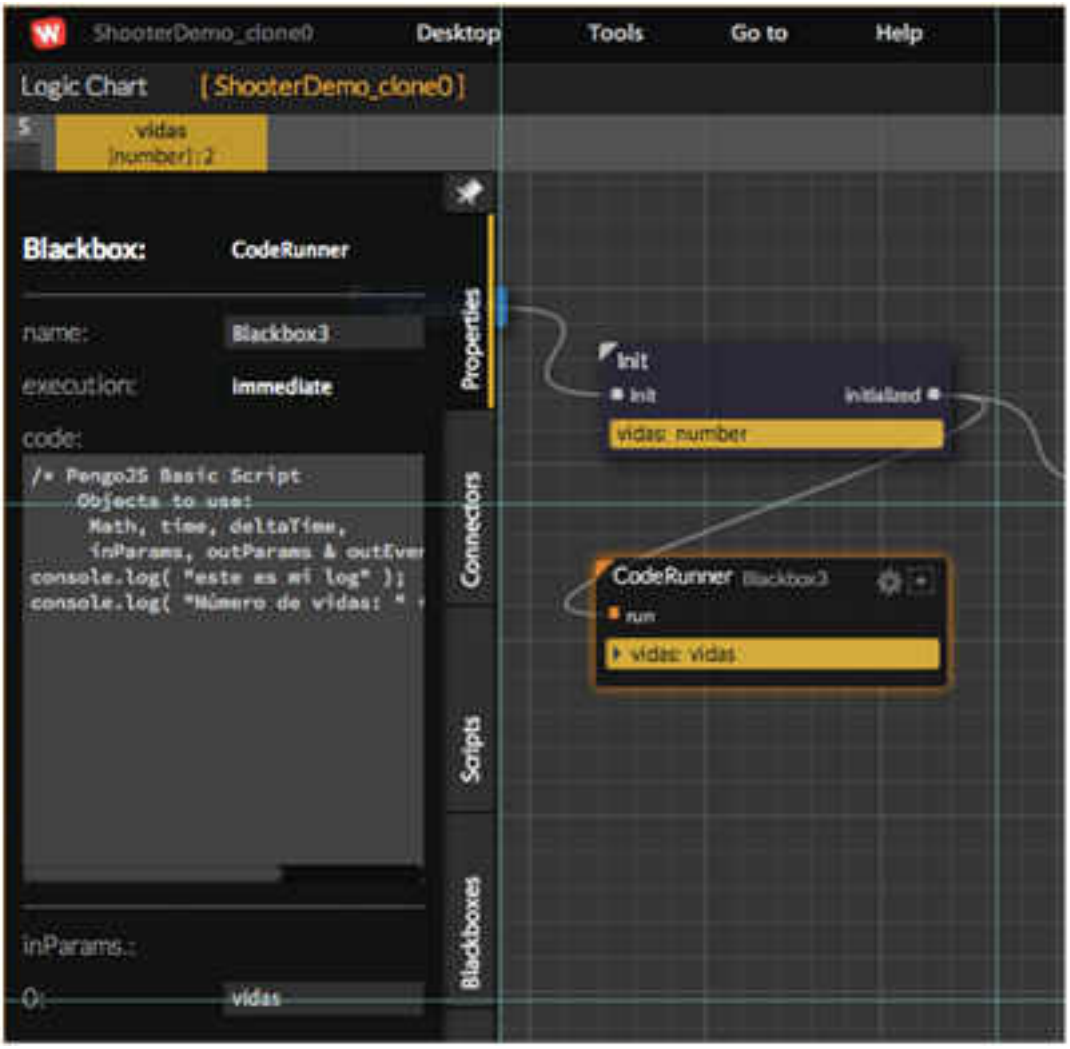

Рисунок 85. Налагодження гри за допомогою чорної скриньки

Пам'ятайте, що цей журнал відображатиметься на консолі налагодження веб-переглядача (рис. 86).

2. З іншого боку, коли ми попередньо переглядаємо гру на панелі попереднього перегляду, ми можемо вибрати будь-який параметр blackbox для перегляду поточного значення цього параметра.

Зазвичай, це значення буде значенням, яке було встановлено в останній раз, коли виконувався Blackbox. Хоча може статися, що цей параметр був посилатися в іншій чорній скриньці, виконаній пізніше, тому це буде значення, яке було встановлено цим параметром.

Це основа для інструменту налагодження, який дозволить розробникам WiMi5 припинити виконання гри в чорно-бічній панелі, щоб ви могли перевірити значення в будь-який час.

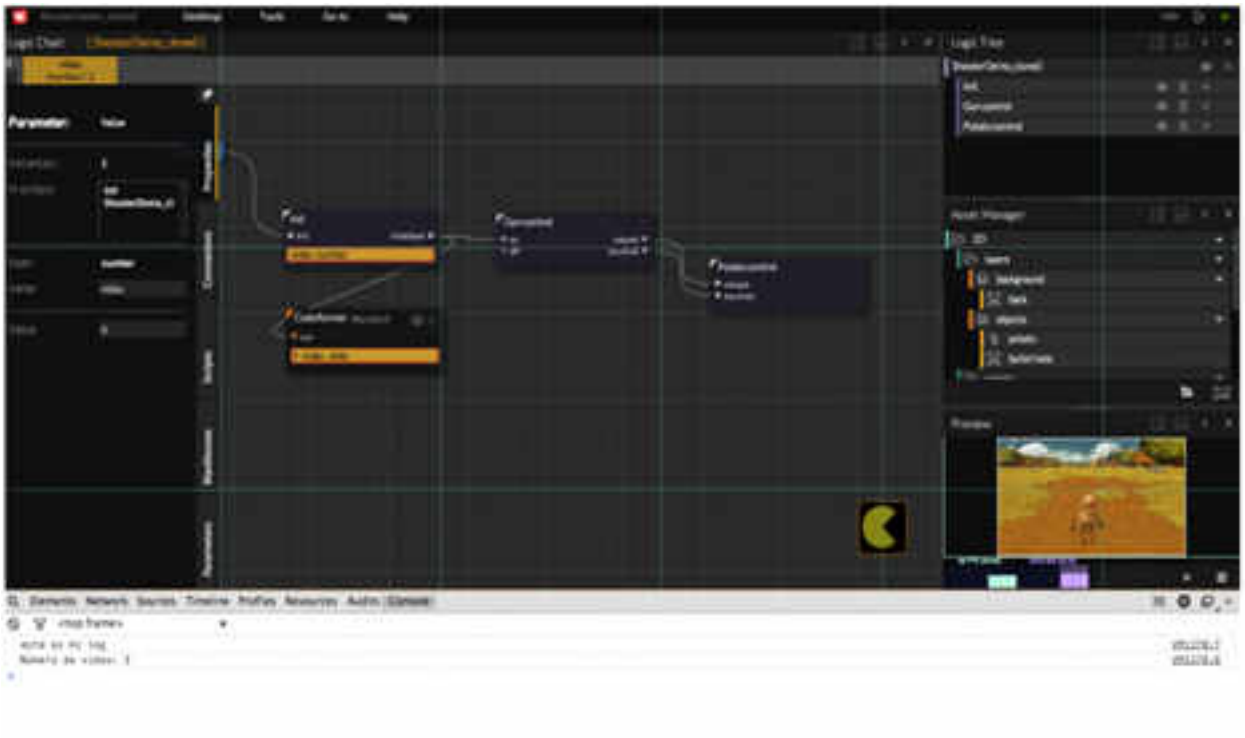

Рисунок 86. Гра Пошук серії слів

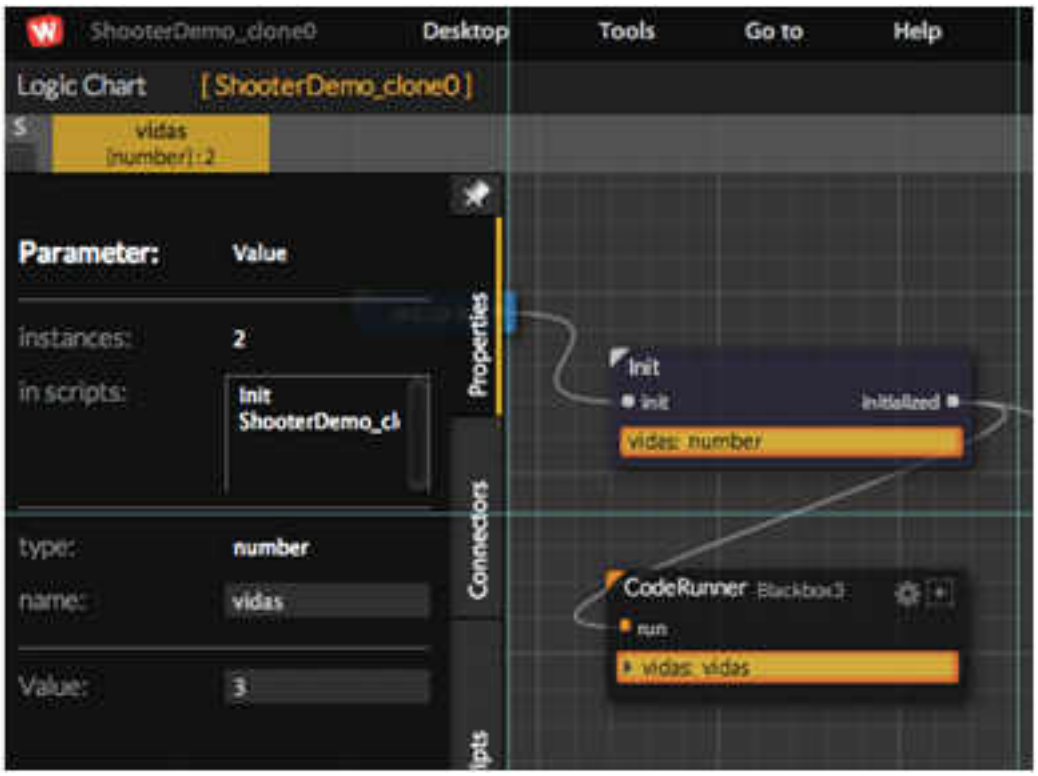

Рисунок 87. Налагодження шляхом попереднього перегляду

ПУБЛІКАЦІЯ ГРИ HTML5 З WIMI5 НА САЙТІ LABS

Якщо ви хочете поділитися експериментами чи проектами HTML5 з іншими розробниками чи гравцями, ви можете опублікувати його на сайті Labs.

Сайт WiMi5 Labs – це відкритий простір для експериментів та створення можливостей, які платформа WiMi5 вільно пропонує спільноті (рис. 88).

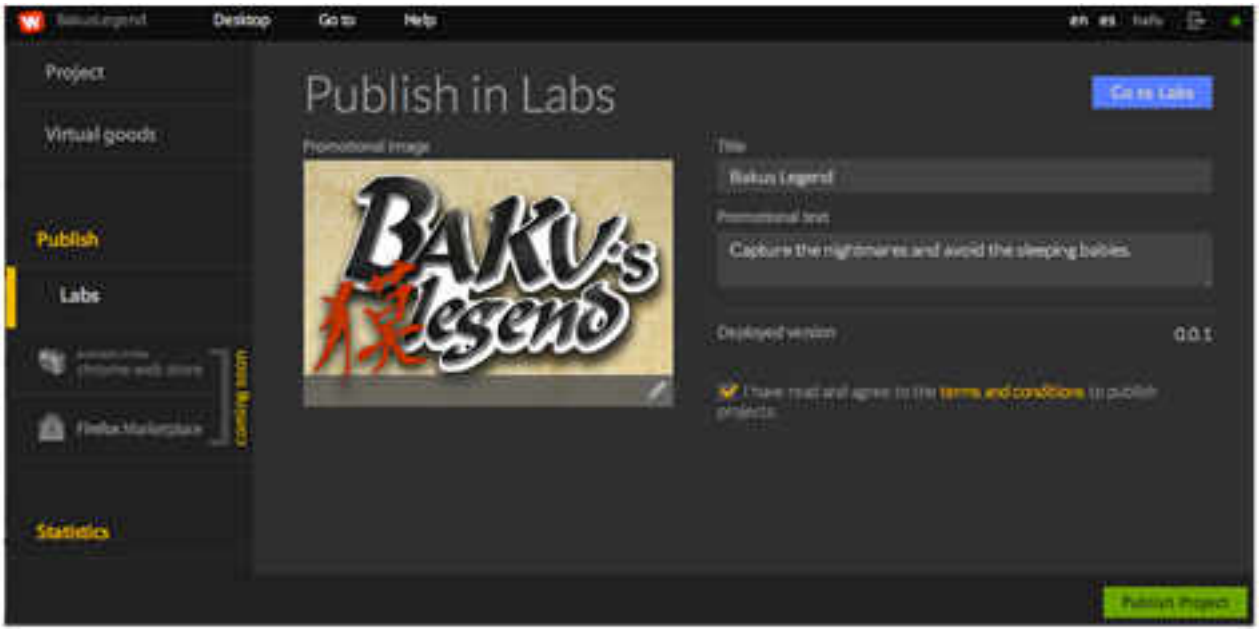

Рисунок 88. Сайт WiMi5 Labs

Щоб опублікувати свою гру, просто натисніть на кнопку «Лабораторії» в лівій частині вікна. У розділі "Публікація в лабораторіях" вам доведеться заповнити деякі відомості про ваш проект: назву, рекламний текст, рекламний малюнок тощо, а потім натисніть кнопку "Опублікувати", щоб зробити її публічною і, отже, доступною для всіх.

Наразі ви зможете публікувати лише в зоні лабораторій, незабаром ви зможете опублікувати його на Game Arena та інших ринках як Chome Web Store або Firefox Marketplace.

- **Назва**: це назва, яку ви хочете показати для вашої гри HTML5 на сайті Labs.
- **Рекламний текст**: це текст, який використовується для опису вашої гри, і буде показано на веб-сайті Labs поруч із заголовком вашого проекту HTML5.
- **Розгорнута версія**: це номер версії, яку ви зараз розгортаєте.
- **Рекламний образ**: цей ескіз відображатиметься на сайті Labs. Необхідна роздільна здатність - 300 × 200 пікселів. Якщо роздільна здатність зображення відрізняється, вона буде масштабована до цієї роздільної здатності. Ви можете завантажити файли .png або .jpg. Щоб вибрати зображення, ви можете скористатись диспетчером об'єктів

(вашим хмарним сховищем в WiMi5) і виділити зображення з вашого каталогу або ви можете завантажити новий.

Після вибору рекламного зображення кнопка "Публікувати проект" буде активною, і ви зможете натиснути на неї та опублікувати свою гру.

Перш ніж це зробити, ви повинні перевірити "Загальні положення та умови" та прийняти їх.

Після опублікування вашої HTML5 гри миттєво буде показано на сайті WiMi5 Labs (рис. 89).

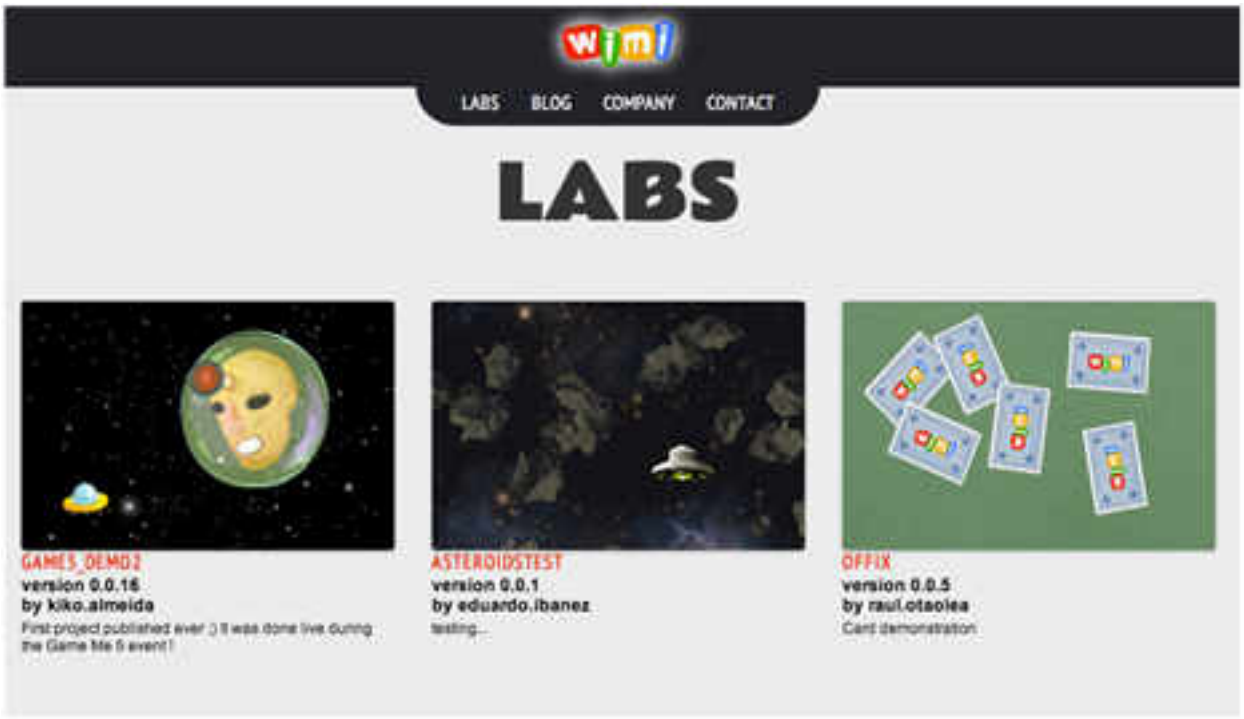

Рисунок 89. Результат публікування гри

WIMI5 GAME ARENA ТА ІНШІ ВИДАВНИЧІ ПАРТНЕРИ

Наразі ви можете публікувати лише в зоні лабораторій, незабаром ви зможете опублікувати його на Game Arena та інших ринках як Chome Web Store або Firefox Marketplace.

ПУБЛІКАЦІЯ ГРИ HTML5 НА ІНШИХ ІГРОВИХ ПЛАТФОРМАХ

Ми збираємося завантажити гру, створену на WiMi5, і ми збираємося її опублікувати на локальному веб-сервері.

Це лише приклад, який дозволить нам перевірити, чи наша гра працює правильно. Але справжньою метою завантаження гри є можливість його опублікування на будь-якому ігровому порталі, щоб зробити її доступною для широкої аудиторії. І пакет, який ви завантажуєте, дозволить це зробити.

СТВОРЮЄМО ЗАВАНТАЖУВАНИЙ ФАЙЛ .ZIP

Перед тим, як упакувати його, потрібно створити та розгорнути вашу гру щонайменше один раз. Пакет, отриманий в результаті цього процесу, - це просто стиснутий .zip, який містить файли, необхідні для запуску на веб-сервері.

1. На інформаційній панелі WiMi5 виберіть один з ваших проектів і перейдіть до налаштувань, натиснувши значок у формі передач. Коли ви побачите подробиці цього проекту, в лівій частині меню з'явиться розділ "Опублікувати" з декількома підрозділами. Клацніть на розділі Завантаження.
2. Створіть .zip-файл своєї гри, натиснувши кнопку "*Пакет*". Цей процес триватиме кілька секунд (рис. 90).

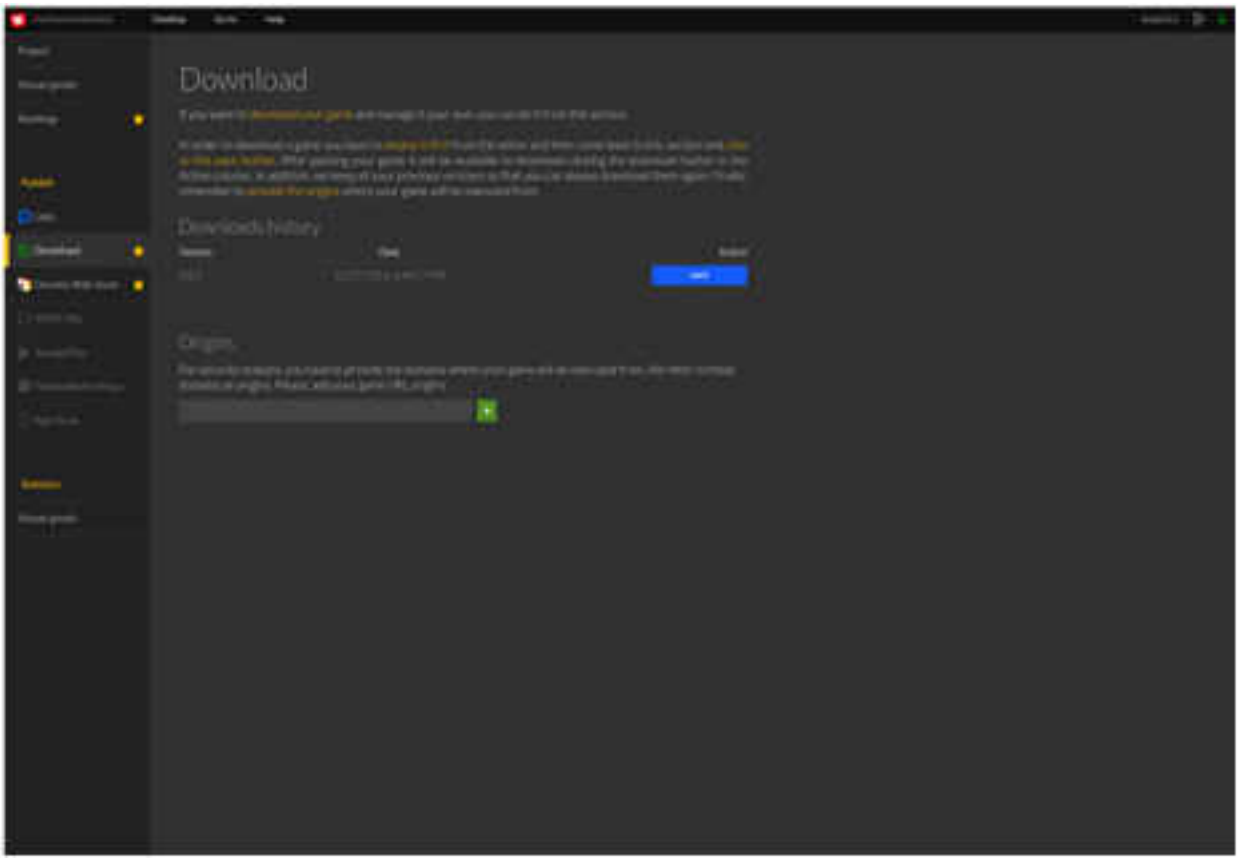

Рисунок 90. Команда «Пакет»

3. Після того, як вона буде упакована, кнопка *пакета* перетвориться на кнопку із піктограмою, яка символізує, що тепер ви можете завантажувати свою гру в якості файлу .zip з файлами вашої гри (рис. 91).

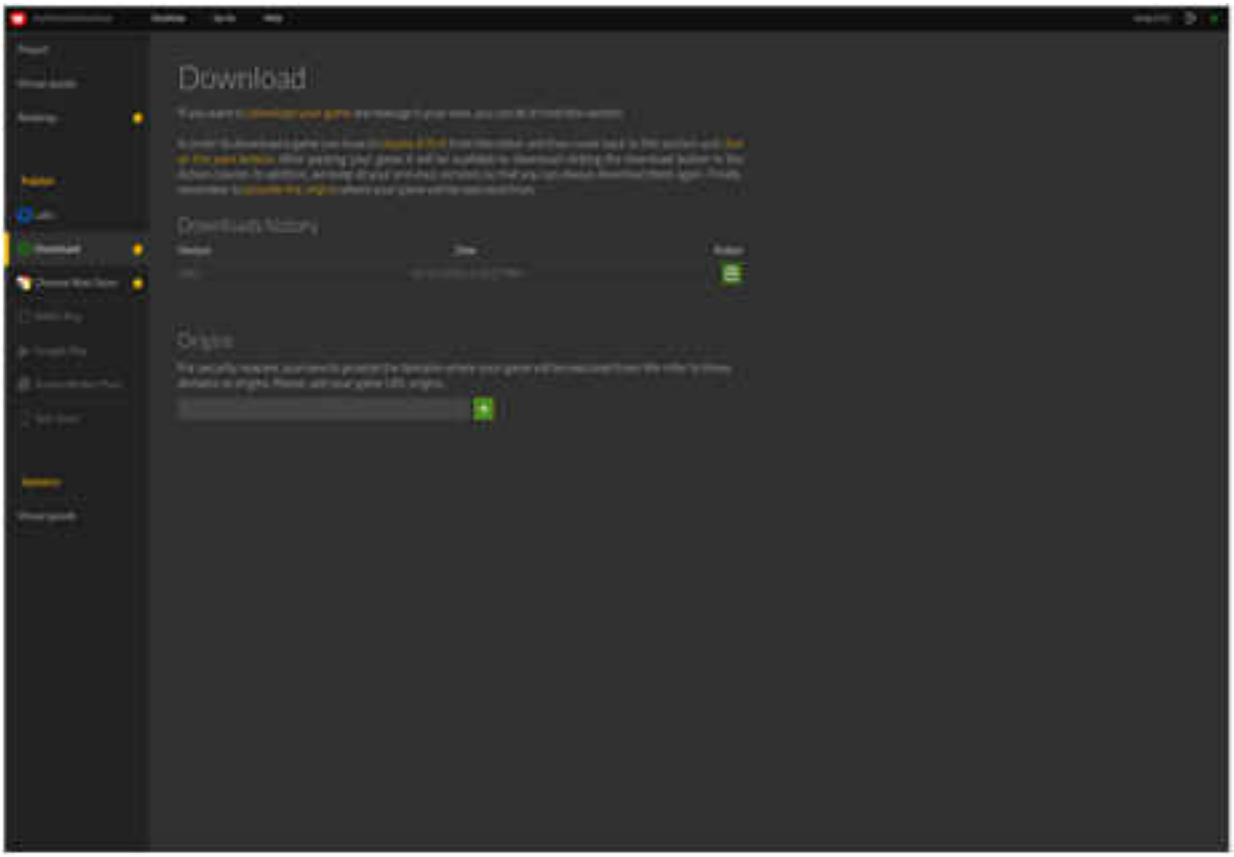

Рисунок 91. Створений zip-файл гри

ПЕРЕВІРЯЄМО, ЧИ ГРА ПРАЦЮЄ

Щоб переконатися, що гра працює правильно, ми будемо запускати її на локальному веб-сервері. Гра має бути авторизована, щоб її можна було запускати з цього конкретного сервера.

Як тільки гра дозволена, ми можемо протестувати гру HTML5 на нашому локальному веб-сервері. Ми повинні мати веб - сервер встановлений на нашому комп'ютері. У цьому прикладі ми будемо використовувати сервер Phyton, але є багато інших веб-серверів.

Давайте подивимося, як перевірити гру на сервері Phyton.

1. Створіть каталог з назвою games.
   mkdir ігри
2. Перейдіть до цього каталогу.
   cd ігри
3. Створіть каталог для гри та перейдіть до нього.
   mkdir myDownloadedApp
   cd myDownloadedApp
4. Розпакуйте гру, завантажену на кроці 1
   розпакувати пакет_myDownloadedApp_0_0_1.zip
5. Перейдіть до каталогу кореневих ігор

cd ..

6. Запустіть веб-сервер. За замовчуванням ігри будуть розміщені на сервері http: // localhost: 8000, написавши це замовлення на консолі: *python-m Простий HTTPServer 8000*
7. Додайте джерело з платформи WiMi5.
8. Відкрийте веб-переглядач і перейдіть за адресою http: // localhost: 8000 / games /
9. Натисніть myDownloadedApp
10. Насолоджуйтесь грою!

АВТОРИЗАЦІЯ ВЕБ-СЕРВЕРА

В даний час необхідно авторизувати веб-сервери, які будуть використовуватися іграми, створеними за допомогою WiMi5.

Це включає локальний веб-сервер, який ми використовуємо в цьому прикладі. Це тому, що ігри, створені за допомогою WiMi5, повинні мати доступ до платформи WiMi5 для використання деяких служб, таких як Управління користувачами, Рейтинги, Постійне зберігання або Віртуальні товари.

Ви повинні перейти на інформаційну панель WiMi5 та вибрати *Налаштування* гри *myDownloadedApp*. Перейдіть до розділу "*Публікація*" і натисніть кнопку "*Завантажити*". У форму *Origins* потрібно ввести URL-адресу вашого веб-сервера. Таким чином ви дозволяєте доступ із цього сервера на платформу WiMi5.

ДОДАЄМО ДОДАТОК ДО ПАНЕЛІ ІНСТРУМЕНТІВ ПРОЕКТУ

Походження має такий формат: <**protocol**>: // <**host**> <**port**>

- **протокол**: це може бути ***http*** або ***https***
- **host**: ім'я хоста (localhost, example.com, google.com, ...) або IP-адреса (127.0.0.1, 142.168.122.136, ...)
- **порт**: порт в діапазоні від 10 до 99999

Наприклад, якщо ми хочемо розмістити нашу гру на сайті Gamejolt, ми повинні використовувати це джерело:

http://www.gamejolt.com

ПУБЛІКАЦІЯ ГРИ HTML5 В ІНТЕРНЕТ-МАГАЗИНІ GOOGLE CHROME

Далі ми розглянемо, як упакувати гру, створену за допомогою WiMi5, і опублікувати її в Інтернет-магазині Google Chrome (CWS), де ваша гра HTML5 буде доступна більш ніж 120 мільйонам геймерів.

Важливо зазначити, що для публікації HTML5-ігор на CWS ви повинні зареєструватися як розробник CWS і сплатити відповідну плату в розмірі 5 доларів США.

КРОК 1: УПАКОВКА ВАШОЇ ГРИ

Перш ніж упакувати гру, потрібно створити і розгорнути свою гру щонайменше один раз.

Пакет, отриманий в результаті цього процесу, є просто compressed.zip, у якому є файли, необхідні CWS.

1. На інформаційній панелі WiMi5 виберіть один з ваших проектів і перейдіть до *налаштувань*, натиснувши значок у формі передач (рис. 92).

Рисунок 92. Панель Налаштування проекту

2. Коли ви побачите подробиці цього проекту, в лівій частині меню з'явиться розділ *"Опублікувати"* з декількома підрозділами. Натисніть розділ *веб-магазину Chrome.* (рис. 93)

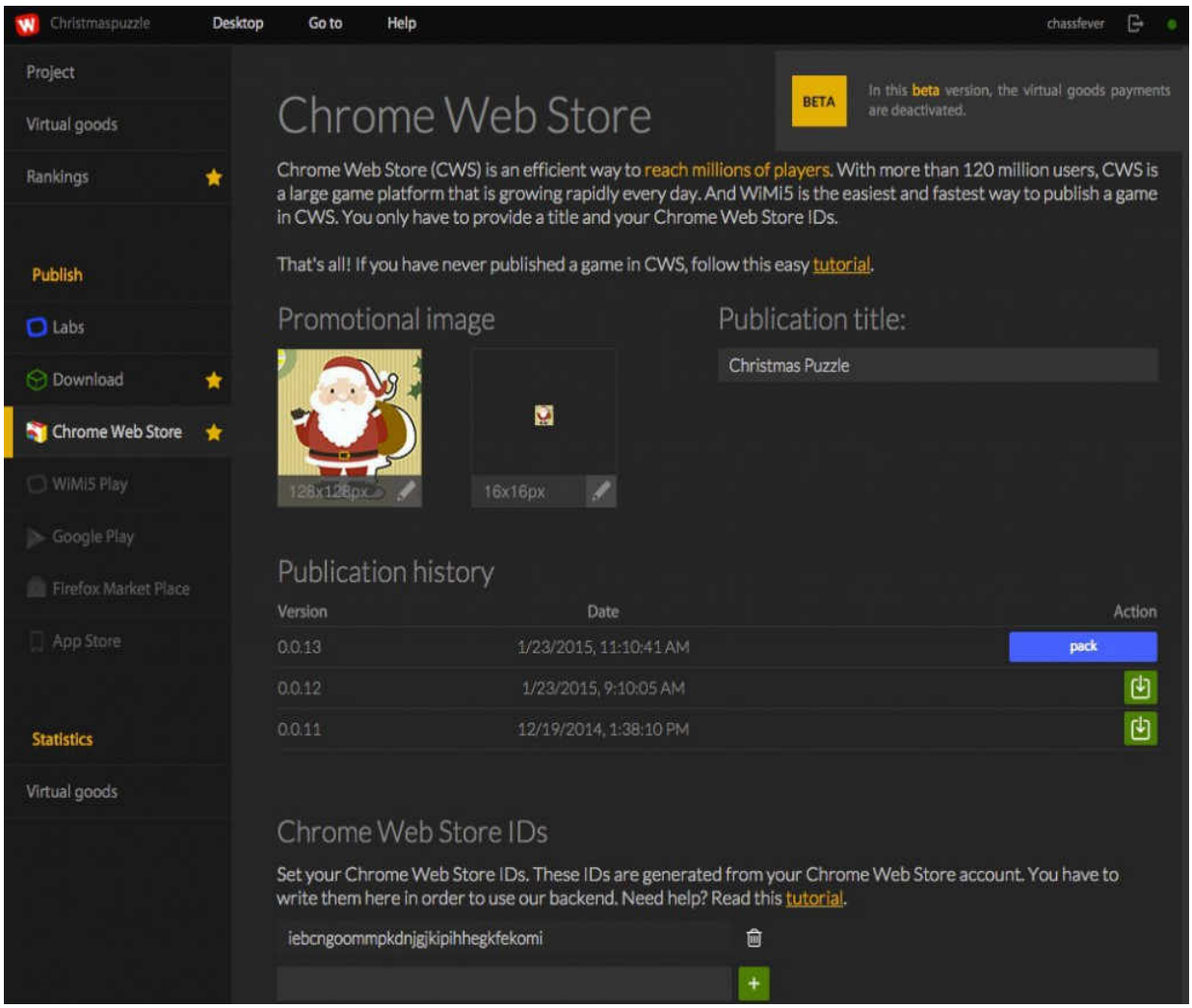

Рисунок 93. Розділ магазину Chrome

3. Важливо дотримуватися послідовності цих кроків. Якщо ви натискаєте кнопку "пакет", перш ніж завантажувати іконки та додавати назву до гри, ці останні дані не будуть включені в файл .zip. Якщо ви хочете змінити іконки або назву, і ви вже упаковали гру, вам доведеться знову розгорнути цю гру. У розділі *Рекламні зображення* завантажте іконки гри. Потім у розділі "*Назва публікації*" введіть назву вашої гри у Веб-магазині Chrome. Цей заголовок повинен містити від 4 до 26 символів.
4. Створіть .zip-файл своєї гри, натиснувши кнопку "Пакет". Цей процес триватиме кілька секунд.
5. Після того, як вона буде упакована, кнопка *пакета* перетвориться на кнопку з піктограмою, яка символізує, що тепер ви можете завантажувати свою гру в якості файлу .zip з файлами, необхідними Веб-магазину Chrome (рис. 94).

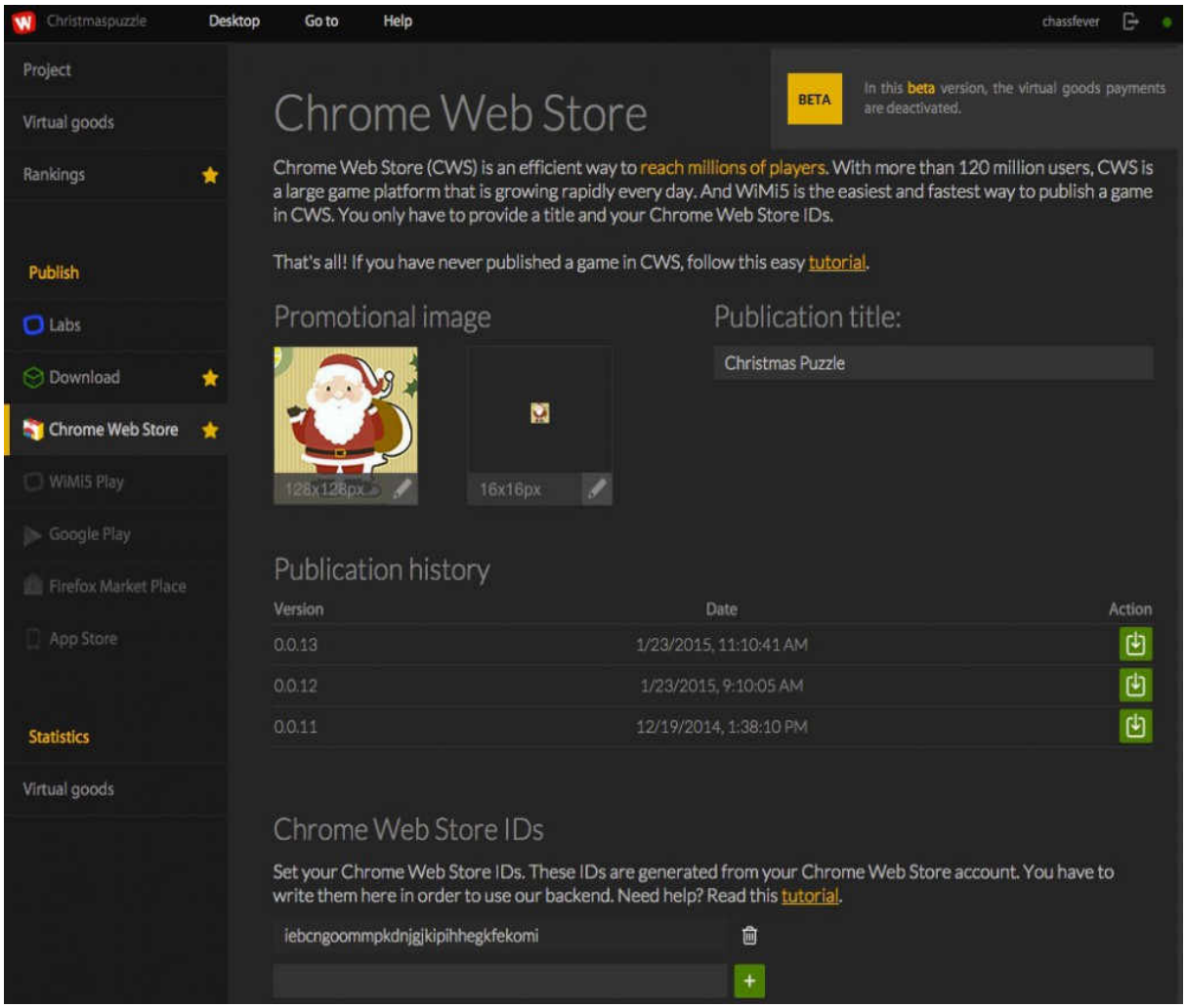

Рисунок 94. Завантаження гри у магазин Chrome

АВТОРИЗАЦІЯ ІДЕНТИФІКАТОРА ДОДАТКА CHROME

Після створення додатка Веб-магазину Chrome нам потрібно знати його ідентифікатор, щоб дозволити його доступ до платформи WiMi5, оскільки всі ігри, створені та упаковані в WiMi5, в даний час використовують служби, такі як керування користувачами, ранжування, віртуальні товари або тривалість ігрових даних.

ОТРИМАННЯ ІДЕНТИФІКАТОРА ДОДАТКА:

Вам потрібно отримати доступ до своєї панелі керування у Веб-магазині Chrome. Натисніть посилання *Більше інформації* для будь-якої вашої гри та ідентифікатор, який ми шукаємо, позначено як ідентифікатор елемента.

АВТОРИЗАЦІЯ ІДЕНТИФІКАТОРА ДОДАТКА НА ІНФОРМАЦІЙНІЙ ПАНЕЛІ WIMI5

Знайшовши ідентифікатор свого додатка, поверніться до розділу "Публікація у *Веб-магазині Chrome*", яку ви хочете опублікувати на *інформаційній панелі* WiMi5.

У розділі "Ідентифікатори веб-магазину Chrome" додайте ідентифікатор додатка. Ваша гра тепер авторизована (рис. 95).

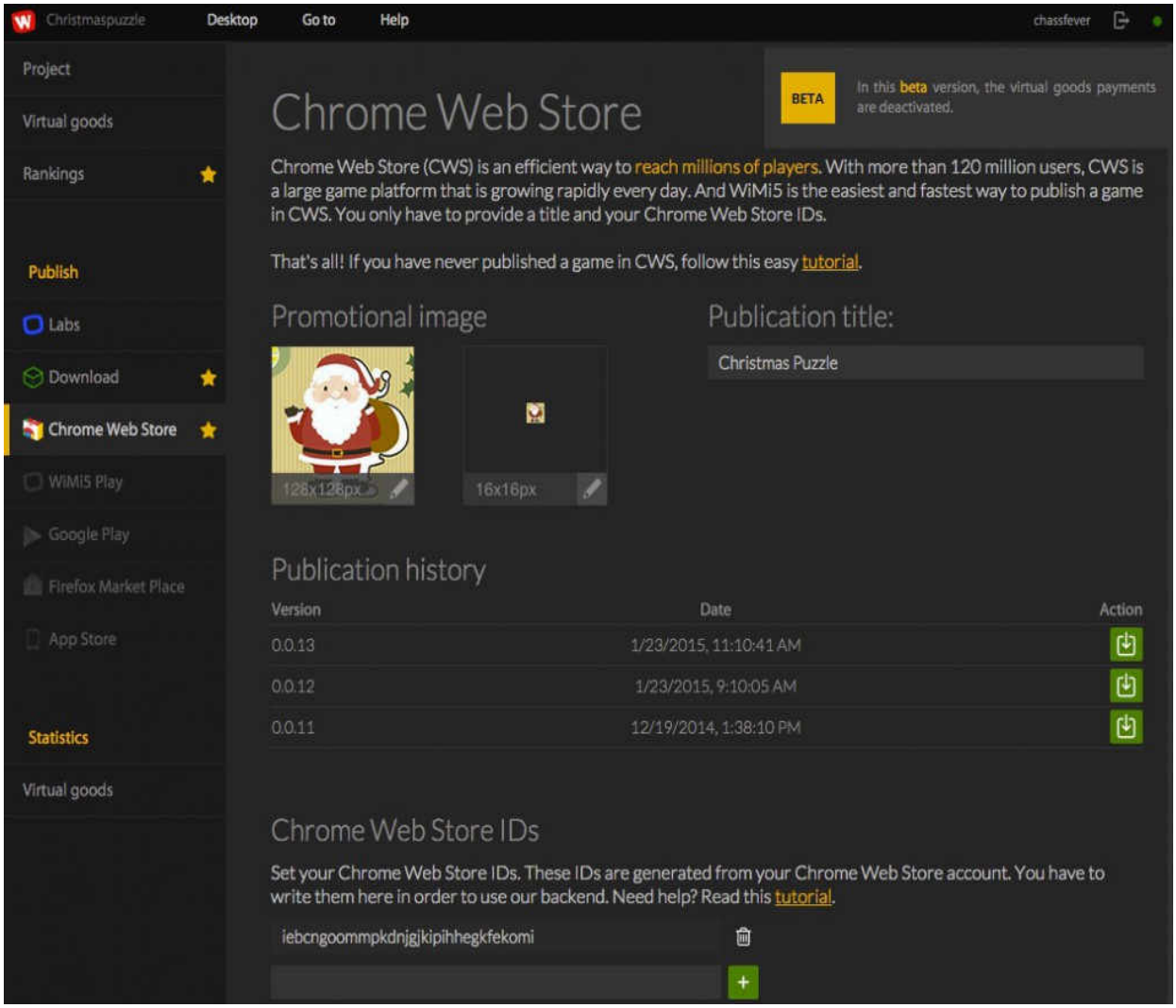

Рисунок 95. Авторизація гри

Лабораторна робота №1
Візуальна розробка елементів ігрового середовища

Анотація.

Лабораторна робота орієнтована на оволодіння студентами навичок візуальної розробки як низькорівневих елементів ігрового середовища (зображення, спрайти, звуки), так і високорівневих елементів (табло з підрахунком балів, ігрова сесія).

Лабораторна робота розрахована на 6 академічних годин і виконується малими робочими групами студентів.

Мета роботи:

Засвоїти навички і прийоми візуальної розробки елементів ігрового середовища засобами інструментарію WIMI5.

Обладнання:

Комп'ютерне обладнання ігрової лабораторії GameLab з встановленим інтернет-броузером Google Chrome.

Хід роботи.

1. Створити проект ігрового додатку за результатами розробки ігрової концепції та ігрової моделі в попередньому навчальному модулі.
2. Визначити низькорівневі елементи ігрового середовища (зображення, спрайти, звуки) які потребують реалізації за проектом ігрового додатку.
3. Розподілити низькорівневі елементи ігрового середовища між членами малої робочої групи.
4. Виконати розробку низькорівневих елементів ігрового середовища за допомогою інструментів WIMI5. Зберегти їх в ігровому проекті.
5. Обговорити між членами малої робочої групи та задати створеним об'єктам ігрового середовища певні властивості.
6. Обговорити між членами малої робочої групи та спроектувати метод підрахунку ігрових результатів для гравців.
7. Спроектувати та реалізувати інструментами WIMI5 табло для відображення ігрового життя та ігрового часу.
8. Протестувати працездатність табло.
9. Спроектувати та розробити метод зберігання сеансу гри та ігрових результатів в локальному та/або хмарному сховищі.
10. Інтегрувати всі проведені розробки до ігрового проекту.
11. Зберегти ігровий проект та опублікувати його в системі дистанційного навчання Moodle.
12. Продемонструвати розроблені елементи ігрового проекту викладачу.
13. Оформити звіт про виконання лабораторної роботи та подати його викладачу через систему дистанційного навчання Moodle.

РОЗРОБКА ТАБЛО ДЛЯ ВІДОБРАЖЕННЯ ЖИТТЯ ТА ЧАСУ

Необхідно дати покрокові пояснення того, як створити табло, яке показує кількість ігрових життів, ігрового часу або ігрових балів, отриманих в відеогрі (рис. 96).

Для того, щоб дати певний контекст, ми використаємо приклад проекту StunPig, в якому можна побачити всі програми, описані в цьому посібнику.

Цей проект може бути клонований з інформаційної панелі WiMi5.

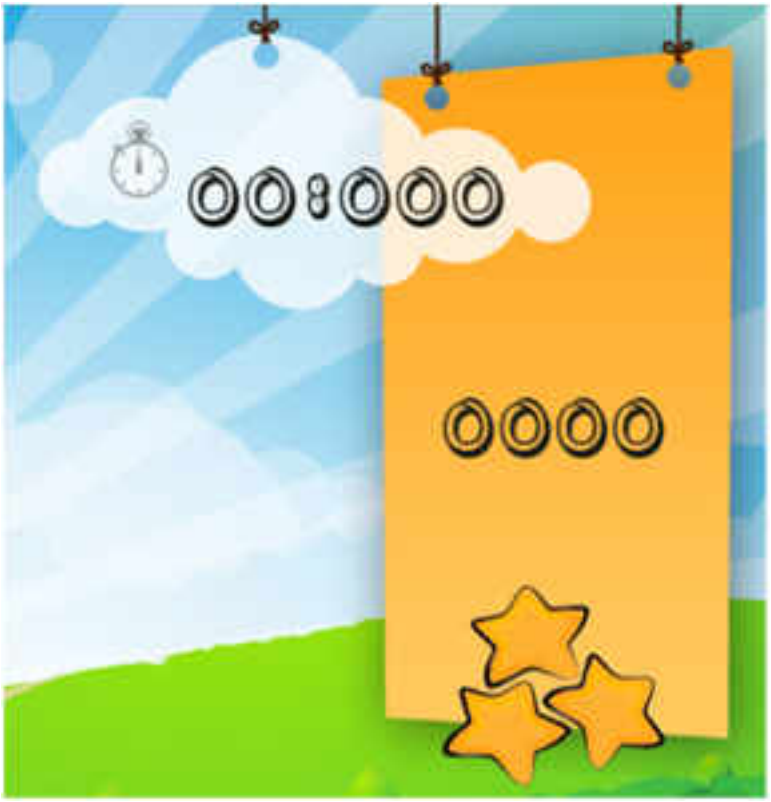

Рисунок 96. Ігрове табло

Для візуалізації значень таблиць необхідні два графічних елементи, анімаційний, що відображає кількість ігрових життєвих обсягів та стільки шрифтів, скільки потрібно для представлення значення цифри, яка буде показана у кожному конкретному випадку.

Спрайт "Життя" – це один спрайт з чотирма анімаціями чи станами зображень, які пов'язані з кожним із чотирьох чисельних значень для значення рівня життя (рис. 97).

Рисунок 97. Графічні елементи спрайту Життя

Шрифт спрайт – це спрайт з 11 анімації або станами зображення, які пов'язані з кожним з десяти значень чисел 0-9, а також додатково один для двокрапки (:) (рис. 98).

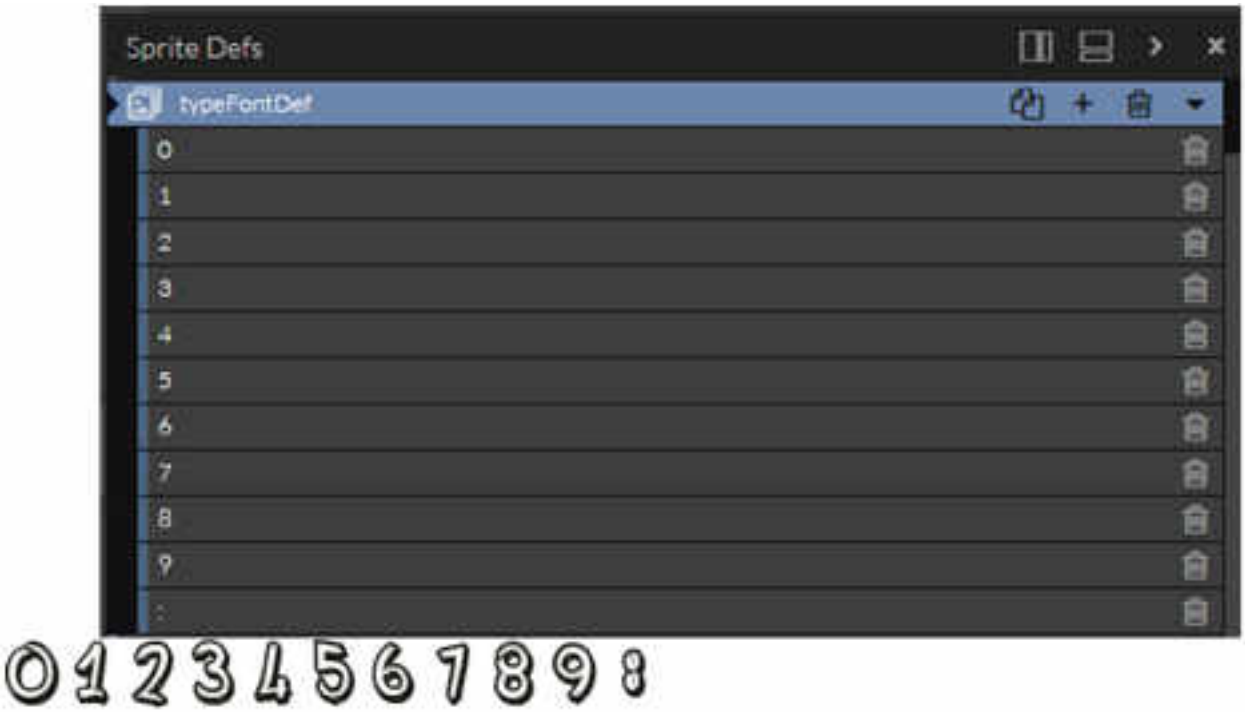

Рисунок 98. Графічні елементи числового спрайту

1. ЯК СТВОРИТИ ТАБЛО ДЛЯ ЖИТТЯ

Щоб керувати життям, для цього знадобиться числове значення, яке в нашому прикладі – це число від 0 до 3 включно, а його графічне зображення в нашому випадку – це три помаранчеві зірки (рис. 99), які змінюються на білі,

коли життя втрачається, та вони мають бути всі білі, коли кількість доступних життів складає 0.

Рисунок 99. Число життів

Для цього в редакторі сцени ми повинні створити екземпляр, який використовується для зірок. У нашому випадку ми називатимемо їх "Життями".

Для маніпуляції ми отримаємо сценарій ("*lifeLevelControl*") з двома входами ("*початок*" і "*зменшення"*), а також двома виходами ("*живий"* і "*мертвий"*) (рис. 100).

Рисунок 100. Сценарій керування життям

Подія "*start*" ініціалізує життя, присвоюючи йому числове значення 3 і починаючи відображення з трьох помаранчевих зірок. Подія "зменшення" знижує числове значення життя на один і відображає відповідні зірки. Як наслідок запуску цієї події, активується один з двох виходів. Вихід "*живого"* активується, якщо після зменшення кількість життя перевищує 0. Вихід "*мертвого*" активується, коли після зменшення кількість життя дорівнює 0.

Усередині сценарію ми робимо все необхідне, щоб змінювати цінність життя, відображати коректний спрайт по відношенню до кількості життя, запускаючи правильний вихід за кількістю життів, а в нашому прикладі також відтворюємо негативний звук несправності, коли кількість життів знижується.

У нашому скрипті «*lifeLevelControl*», у нас є параметр «*currentLifeLevel»*, який містить кількість життів, і параметр, який містить спрайт «*життя»*, який є елементом екранного відображення, яке представляє життя (рис. 101). Цей спрайт має чотири анімації станів: "0", "1", "2" і "3".

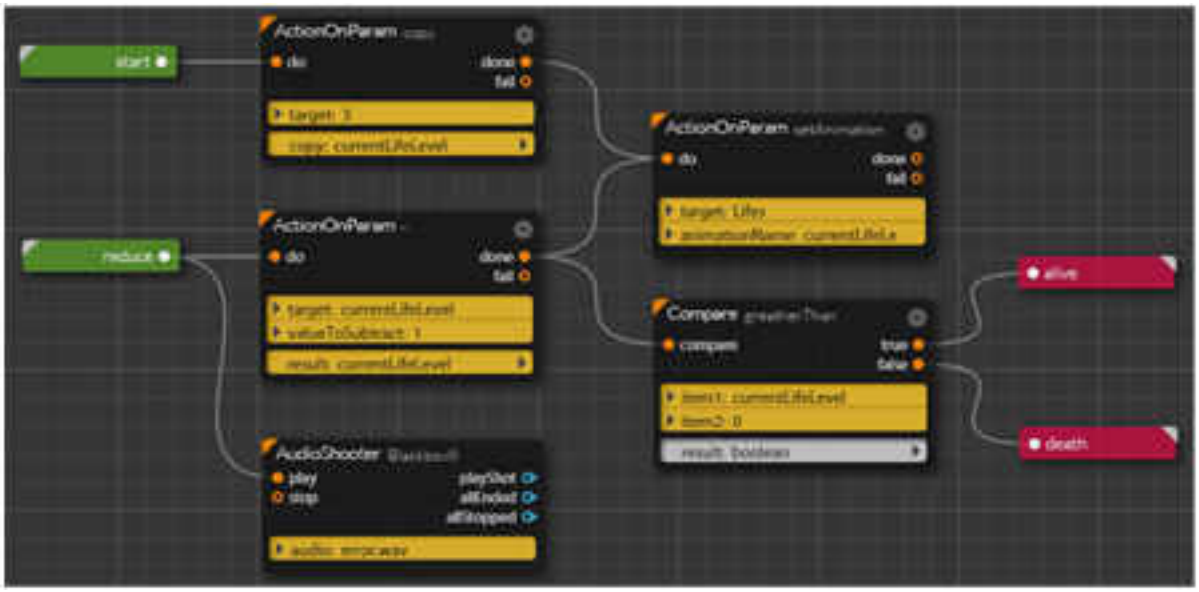

Рисунок 101. Гра Пошук серії слів

Вхід "*start*" активує *Blackbox* копію *ActionOnParam,* яка присвоює параметру *"currentLifeLevel"* значення 3, після чого активується *BlackBox ActionOnParam "setAnimation",* який відображає анімацію спрайту "3".

Вхід «*Зменшити*» активує «-» *ActionOnParam Blackbox*, який віднімає від значення параметру «*currentLifeLevel»* 1. Як тільки це зроблено, він спочатку активує в *Blackbox ActionOnParam* «*setAnimation»*, який відображає анімацію або стан, що відповідає значенню параметру "*CurrentLifeLevel",* а по-друге, він активує *"bigThan"* в *Compare Blackbox*, який активує *"живий"* коннектор, якщо значення параметра *"currentLifeLevel"* більше 0, або "*мертвий"* коннектор, якщо значення дорівнює або менше 0.

2. ЯК СТВОРИТИ ТАБЛО АБО ХРОНОМЕТР ЧАСУ

Щоб керувати часом, ми матимемо чисельне значення часу, яке буде обчислюватись у тисячних долях секунди в раунді, та графічний елемент для його відображення. Цей графічний елемент буде мати 5 екземплярів спрайту, кожний з яких буде мати 10 анімацій або станів, які будуть цифрами від 0-9 (рис. 102).

У нашому випадку ми покажемо час у секундах і тисячних секундах, який ви можете бачити на зображенні, підраховуючи час; так що час буде йти від загального часу на початку і знижатися до досягнення нуля.

Для цього в редакторі сцен ми повинні створити 6 екземплярів різних спрацьовувань, які використовуються для кожного сегмента дисплея часу. У нашому випадку ми називатимемо їх "*second.ten"*, "*second.unit"*, *"millisec.unit*", "*millisec.ten*" та "*millisec.hundred*" (рис. 103).

Рисунок 102. Хронометр часу

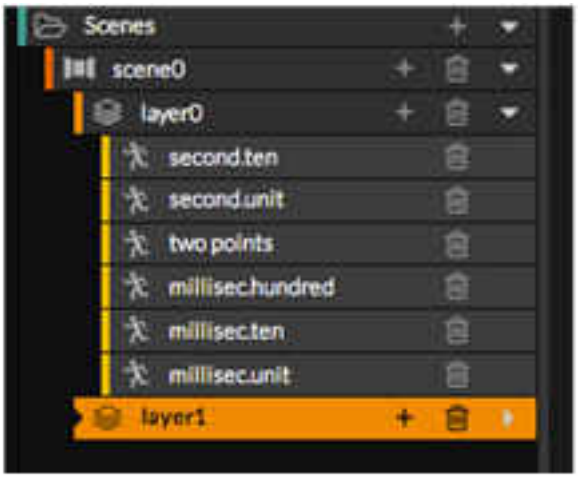

Рисунок 103. Види спрацьовувань

Щоб керувати цим часом, ми створюємо сценарій ("*RoundTimeControl"*), який має 2 входи ("*початок*" і "*зупинка*") і 1 вихід ("*кінець*"), а також відкритий параметр "*roundMillisecs*" і який містить значення часу початку.

Рисунок 104. Сценарій хронометрування

Вхід "*start"* активує зворотний відлік від загального часу і відображає зменшуване значення в секундах і мілісекундах.

Вхід "*stop"* зупиняє зворотний відлік, заморожуючи поточний час на екрані.

Коли закінчиться встановлений час, активізується вихід "*кінець*", який визначає, що час закінчився.

Всередині сценарію ми робимо все, що потрібно для керування часом і відображенням спрайтів у співвідношенні із значенням часу, що залишився, активізуючи вихід "*кінець",* коли він таки закінчиться.

Для того, щоб використовувати цей сценарій, все, що нам потрібно зробити, це поставити значення часу у мілісекундах або шляхом розміщення його безпосередньо в параметрі "*roundMillisecs"* або за допомогою *чорної скриньки*, призначити його (рис. 105), і після цього потім активувати вхід "*start"*, який показуватиме зворотний відлік, поки ми не активізуємо вхід "*stop"* або не досягнемо 0, у такому разі буде активовано вихід "*кінець*", який ми можемо використовувати далі, наприклад, для активації видалення життя чи чогось іншого.

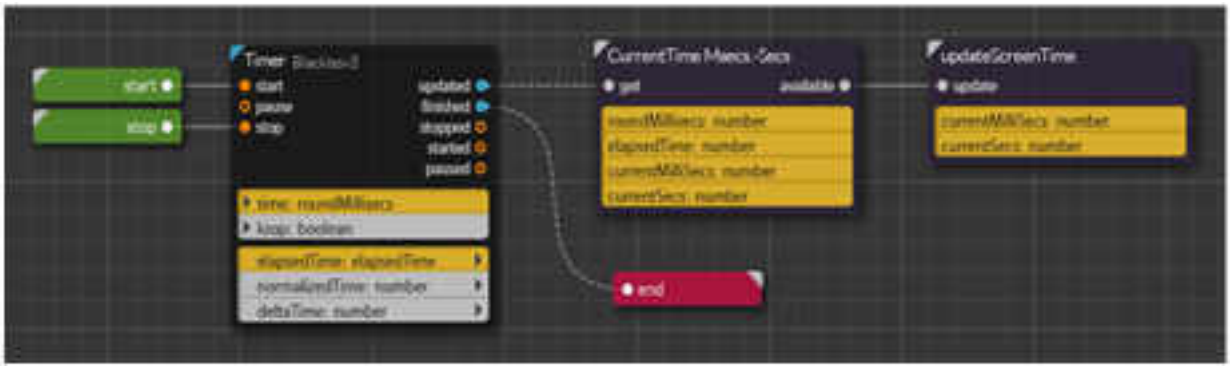

Рисунок 105. Налаштування сценарію хронометрування

У *скрипті* "*RoundTimeControl" у* нас є фундаментальний параметр "*roundMillisecs",* який містить і визначає значення часу відтворення в раунді.

Всередині цього сценарію ми також маємо ще два скрипти, "*CurrentMsecs-Secs"* та "*updateScreenTime"* , які об'єднують дії, які описано нижче.

Активація з'єднувача "*start*" активує вхід "*start*" таймера *BlackBox*, який починає зворотний відлік. Оскільки визначений час відраховується, цей *BlackBox* оновлює параметр "*пройденого часу"*. Це відбувається з самого першого моменту і повторюється до останнього моменту перевірки часу, коли спрацьовує вихід «*кінець»*, повідомляючи про те, що час закінчився. З огляду на те, що час запуску не повинен бути кратним часу між оновленням і перевіркою часу, остаточне значення параметру *минулого часу*, швидше за все, буде більшим за вимірювання, що потрібно мати на увазі, коли це необхідно.

Вихід *«Оновлено»* говорить нам , що ми маємо нове значення в параметрі «*ElapsedTime»* і активувати сценарій «*CurrentTimeMsecs-ІКС»*, який обчислює загальний час, що залишився в мілісекундах і ділить його в секунди і мілісекунди, щоб відобразити. Після того, як ця інформація отримана, вихід «*доступно»* буде спрацьовувати, що в свою чергу активує вхід «*оновлення»*

сценарію «*updateScreenTime*», який поміщає відповідні анімації в спрайти, що відображають час.

У *сценарії* "*CurrentMsecs-Secs*" ми маємо два фундаментальні параметри; "*RoundMillisecs*", який містить та визначає значення часу відтворення в раунді та "*expiredTime*", який містить кількість часу, що минув з початку роботи годинника (рис. 106). У цьому сценарії ми обчислюємо час, що залишився, і тоді ми розбиваємо цей час у мілісекундах на секунди і мілісекунди, останнє виконується в сценарії "*CalculateSecsMillisecs*".

Рисунок 106. Сценарій поточного часу

Активація роз'єму *GET* починає обчислення часу, що залишився, починаючи з активації «-» *ActionOnParam Blackbox*, що віднімає значення часу, який минув з моменту присвоєння параметру «*ElapsedTime*» загального значення часу виконання, що міститься в параметрі "*roundMillisecs*". Це значення, яке зберігається в параметрі "*CurrentTime*", - час, що залишився в мілісекундах.

Після того, як це було обчислено, *активізується порівняння BlackBox* "*GreaterOrEqual*", який порівнює значення, що міститься в "*CurrentTime*" (час що залишився), до значення 0 (рис. 107). Якщо він більше або дорівнює 0, він активує сценарій "*CalculateSecsMillisecs*", який розбиває залишковий час на секунди і мілісекунди, і коли це зроблено, він запускає вихідний роз'єм "*доступно*". Якщо ж менше, перед активацією сценарію "*CalculateSecsMillisecs*" ми активуємо копію *Blackbox ActionOnParam,* яка встановлює значення, що залишилось, в нуль.

Рисунок 107. Порівняння часу

У сценарії "*CalculateSecsMillisecs"* ми маємо значення часу, що залишився в мілісекундах, що міститься в параметрі "*currentTime"* як вхідний. Сценарій розбиває це вхідне значення на його значення в секундах і його значення в мілісекундах, надаючи їм параметри "*CurrentMilliSecs" та "CurrentSecs"*. Активація вхідного роз'єму "*get"* активує *"lessThan"* в *Compare blackbox*. Це виконує порівняння значення, що містяться в параметрі "*currentTime",* щоб побачити, чи це менше 1000.

Якщо це менше, то вихід "*справжній"* спрацьовує. Це означає, що немає секунд, тобто ціле значення "*CurrentTime"* використовується як значення в параметрі "*CurrentMilliSecs"*, який потім копіюється *BlackBox ActionOnParam* "*Copy"*; але він не копіює секунди, тому що вони 0, і це дає значення нуля для параметра "*currentSecs"* через *BlackBox ActionOnParam* "*copy"*. Після цього він має значення, надані сценарієм, так що він активує свій "*завершений"* вивід.

З іншого боку, якщо прапорець «*LessThan» Compare* BlackBox працює визначає, що «*CURRENTTIME»* більше, ніж 1000, він активізує свою «*помилковий»* висновок. Це активує Blackbox *BlackBox ActionOnParam* , який розподіляє параметр "*currentTime"* на 1000', зберігаючи його в параметрі *totalSecs* . Після цього активується "*floor" ActionOnParam* , що залишає його загальну суму "*totalSecs"* у параметрі "*currentSecs"* .

Після цього активується "-" *ActionOnParam*, який віднімає "*currentSecs"* від "*totalSecs",* що дає нам десяткову частину "*totalSecs"* і зберігає його в "*currentMillisecs"*, щоб пізніше активізувати "*" *ActionOnParam* blackbox, *множивши* на 1000 параметр "*currentMillisecs",* який містить десяткове значення секунд, що залишилося, для перетворення його в мілісекунди, яке зберігається в параметрі "*CurrentMillisecs"* (видалення попереднього значення). Після цього він має значення, надані сценарієм, після чого він активує його "*завершений"* вивід.

Коли обробка "*CalculateSecsMillisecs"* закінчується та активізується, виводиться "*завершений*", і це активує "*доступний"* вивід сценарію, активується сценарій "*currentTimeMsecs-Secs"*, який потім

активує сценарій "*updateScreenTime"* за допомогою введення "*update"*. Цей сценарій обробляє дані, отримані в попередньому сценарії, і які доступні в параметрах "*CurrentMillisecs"* та "*CurrentSecs"*.

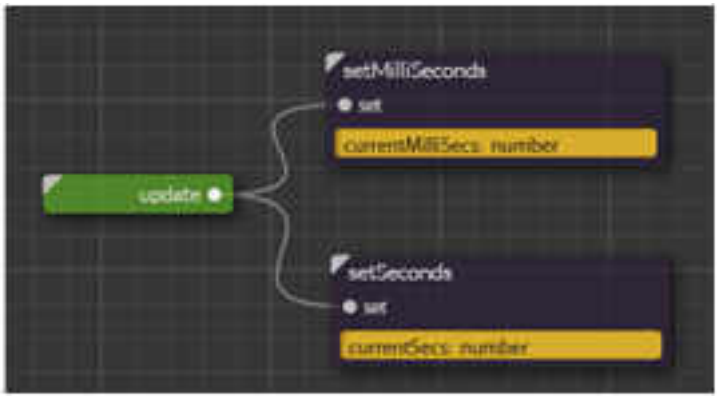

Рисунок 108. Логіка сценарію зміни часу

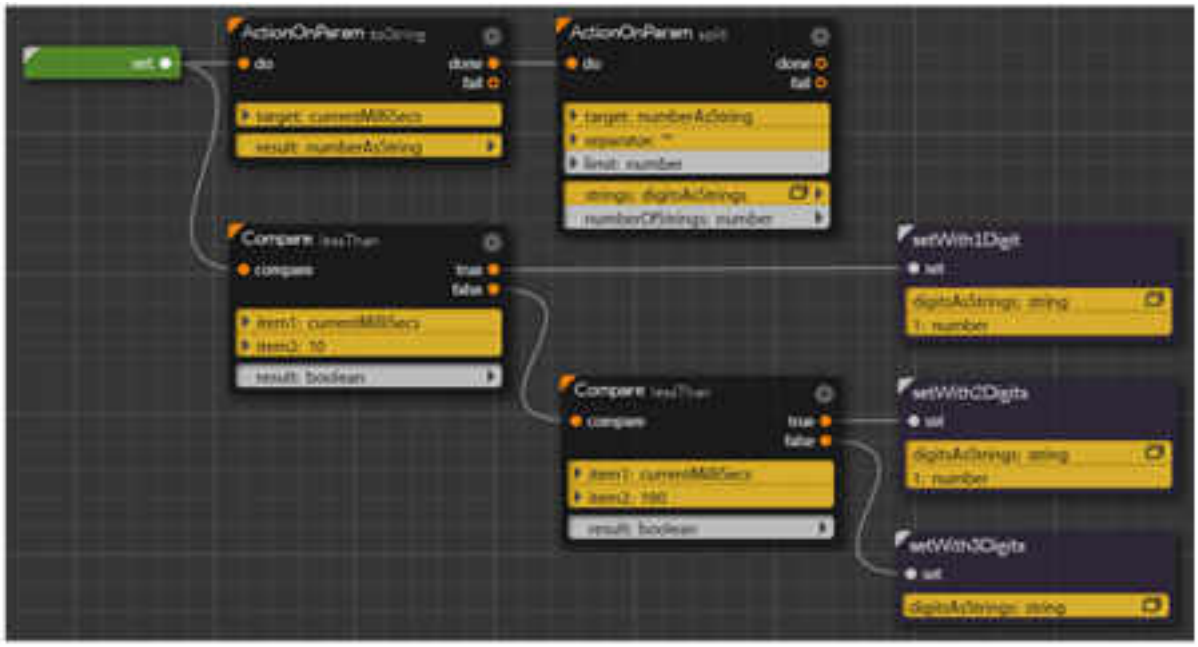

Рисунок 108. Логіка сценарію встановлення часу

Сценарій " *updateScreenTime",* у свою чергу, містить два скрипти, "*setMilliSeconds* " і "*setSeconds*", які активуються при ввімкненні введення " *update"* і які встановлюють значення часу у мілісекундах і секундах відповідно, коли активізуються вхідні "*set* ". Обидва скрипти практично однакові, оскільки вони приймають значення часу і розміщують Sprites, пов'язані з одиницями цього значення у відповідних анімаціях. Різниця між двома *полями* полягає в тому, що "*setMilliseconds"* контролює 3 цифри (десятих, сотих і тисячних), а "*setSeconds"* контролює лише 2 одиниці та десятки.

Перше, що «*setMilliseconds*» Скрипт робить, коли активується, це перетворити значення «*currentMillisecs*» представлятиме в текст з допомогою «*ToString*» *ActionOnParam* Blackbox. Цей текст зберігається в параметрі "*numberAsString*". Після того , як текст був отриманий, ми розділимо його на персонаж, групуючи її в колекції рядків через «*спліт*» *ActionOnParam*. Це дуже важливо, щоб залишити зміст «*роздільник*» параметра цього чорного ящика порожній, навіть якщо в зображенні ви можете побачити дві лапки в поле. Ця колекція персонажів збирається за допомогою "*digitsAsStrings*"параметр Пізніше, в залежності від значення мілісекунд, яке буде представлено, він буде встановлювати одну анімацію або іншу в Sprites.

Якщо значення часу, яке буде представлено, має бути меншим, ніж 10, яке перевіряється *"lessThan" Compare* blackbox проти значення 10, активізується "true" вивід, який, у свою чергу, активує сценарій "*setWith1Digit*" (рис. 109). Повинно бути значення часу більше, ніж 10, «в Blackbox в *брехня*» вихід активується, і він переходить перевірити, якщо значення часу менше, ніж 100, який перевіряється з допомогою «*LessThan*» *Порівняти* Blackbox зі значенням 100. Якщо BLACKBOX активізує його "*справжній*" висновок, це, у свою чергу, активує сценарій "*setWith2Digits*". Нарешті, якщо ця *чорна скринька* активує "*помилковий*" вивід, сценарій "*setWith3Digits*" активується.

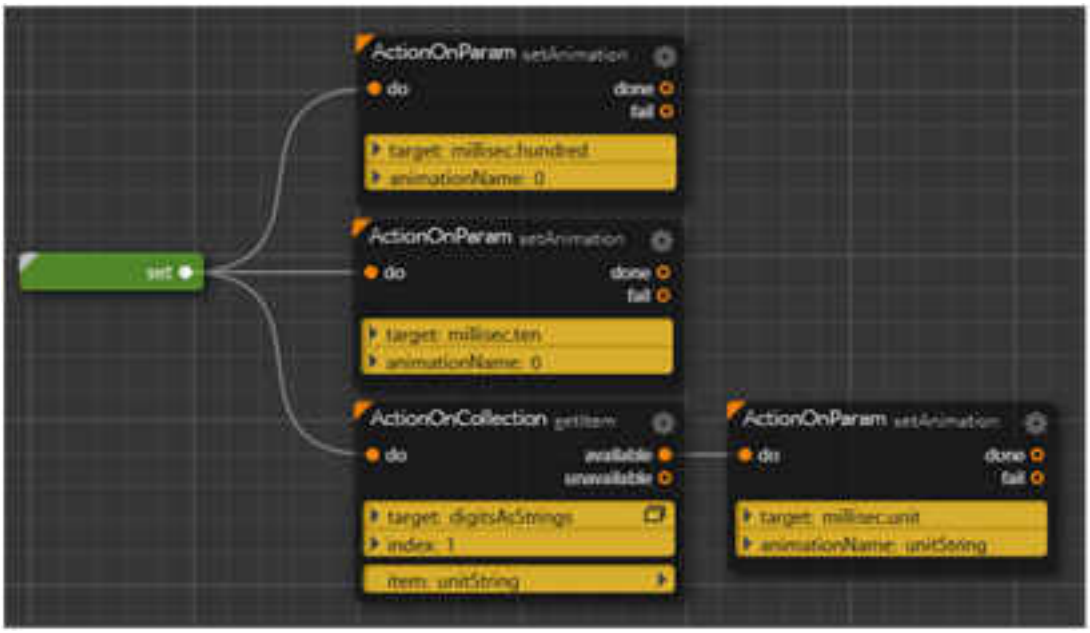

Рисунок 109. Підрахунок мілісекунд

Сценарій "*setWith1Digit*" бере першу з колекції символів (рис. 110) і використовує її для встановлення анімації Sprite, яка відповідає одиницям, що містяться в параметрі "*millisec.unit*". Решта спрайтів («*Міллісек.Десять*» і «*Міллісек.Сто*») встановлені з 0 анімації.

Рисунок 110. Встановлення анімації одиниць мілісекунд

Сценарій "*setWith2Digits*" (рис. 111) бере першу з колекції символів і використовує її для встановлення анімації Sprite, що відповідає десятиному номеру місця, що міститься в параметрі "*millisec.ten"*, другий символ колекції для встановлення Sprite анімація, відповідна одиницям, що містяться в параметрі "*millisec.unit",* і "*millisec.hundred"* Sprite надається анімація для 0.

"*SetWith3Digits*" Sprite приймає першу з колекції символів (рис. 112) і використовує її для встановлення анімації Sprite, що відповідає сотам, що містяться в параметрі "*millisec.hundred"*, другий символ колекції для встановлення анімації Sprite, що відповідає значенню десятих значень, що містяться в параметрі "*millisec.ten"*, і третій символ колекції, щоб встановити анімацію Sprite, яка відповідає значенню місця розташування одиниць, що міститься в параметрі "*millisec.unit"*.

Сценарій "*setSeconds*" при першій активації перетворює значення, яке представляє "*currentSecs",* для тексту через Blackbox *BlackBox ActionOnParam* "*toString"*. Цей текст згруповано в параметрі "*numberAsString"*. Як тільки текст буде отримано, ми розділяємо його на символи, збираючи його в колекцію Strings через "*роздільну" ActionOnParam* blackbox. Дуже важливо залишити зміст параметра "*сепаратор*" цього чорного ящика порожнім, навіть якщо ви можете побачити два лапки в полі. Ця колекція персонажів збирається в "*digitsAsStrings"* параметр Пізніше, на основі показника секунд, який буде показано, одна анімація або інша буде розміщена у Sprites.

Рисунок 111. Встановлення анімації десятків мілісекунд

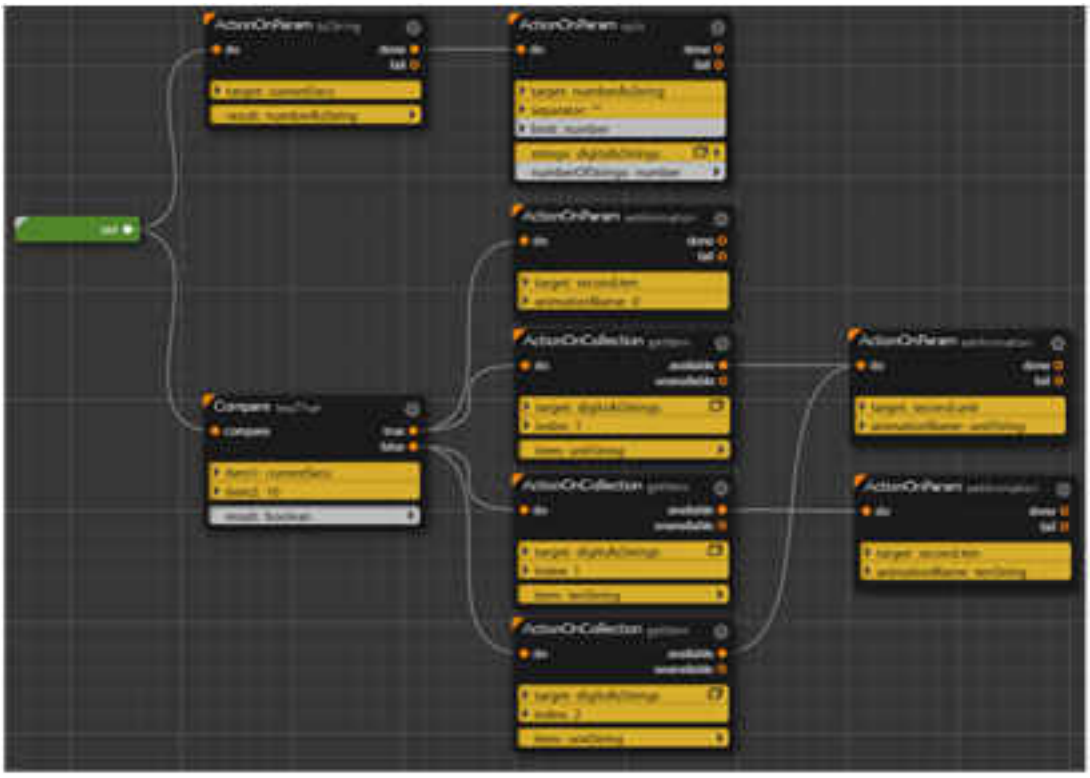

Рисунок 112. Встановлення анімації сотень мілісекунд

Якщо значення часу, яке має бути представлене, менше 10, його перевіряє *"lessThan" Compare* blackbox проти значення 10, яке активує "*true"* вивід; перший символ колекції знімається і використовується для встановлення анімації Sprite, що відповідає значенню місця розташування одиниць, що містяться в параметрі "*second.unit"*. Інший Sprite, "*second.ten"*, надається анімація для 0.

Якщо значення часу, яке буде представлено, перевищує десять, активується «*помилковий»* вивід чорного ящика, і він продовжує вибирати перший символ із набору символів, і ми використовуємо його для встановлення анімації Sprite, яка відповідає що міститься в параметрі "*second.ten"*, а другий символ колекції символів використовується для встановлення анімації Sprite, яка відповідає значенню місця розташування одиниць, що міститься в параметрі "*second.unit"*.

3. РОЗРОБКА ТАБЛО ІГРОВИХ БАЛІВ

Для того, щоб керувати кількістю балів, ми матимемо базу цілих значень цих точок, які ми будемо збільшувати, і графічним елементом для його відображення (рис. 113). Цей графічний елемент буде 4 примірниками Sprite, який буде мати 10 анімацій або станів, кожен з яких буде складатися з 0 до 9.

Рисунок 113. Елементи табло балів

У нашому випадку ми покажемо бали до 4 цифр, тобто кількість балів може досягати 9999 (рис. 114), як ви можете бачити на зображенні, починаючи з 0, а потім збільшуючи цілі числа. .

Рисунок 114. Табло балів

Для цього в редакторі сцени ми повинні створити чотири екземпляри різних Sprites, які використовуються для кожного з чисельних одиниць, які будуть використовуватися для підрахунку балів: одиниць, десятків, сотень і тисяч. У нашому випадку ми називатимемо їх "*одиничною точкою*", "*десяти пунктом*", " *то пункту*" і "*тисяч точок*".

Щоб керувати цим часом, ми отримаємо сценарій ("*ScorePoints"*), що має 2 входи ("*скидання*" та "*приріст*"), а також відкритий параметр "*pointsToWin*", який містить значення точок, які повинні бути додано в кожному збільшенні.

Рисунок 115. Сценарій підрахунку балів

Вхід "*reset"* встановлює поточне значення балів до нуля, а вхід "*increment"* додає значення, виграні при кожному збільшенні, що містяться в параметрі *"pointsToWin",* до поточної оцінки.

Для того, щоб використовувати його, ми повинні встановити лише значення для очок, щоб виграти в кожному збільшенні, або *вставивши* його в параметр "*pointsToWin"* або використовуючи *чорний ящик,* який я його призначаю. Як тільки я його отримаю, ми зможемо активувати вхід "*збільшення"*, що збільшить оцінку та покаже її на екрані. Кожного разу, коли ми хочемо, ми можемо почати знову, скинувши лічильник на нуль, активізувавши введення "*скидання"*.

Усередині Script ми робимо все необхідне, щоб виконати ці дії (рис. 116), а також представляти поточний бал на екрані, відображаючи 4 значення Sprites (одиниці, десятки, сотні та тисячі) стосовно цього значення. Коли активовано вхід "*reset"*, *BlackBox ActionOnParam копія* встановлює значення в 0 в параметрі "*scorePoints"*, яке містить значення поточного балу. Також, коли активовано вхід "*приріст"*, *червона панель* "+" *ActionOnParam* додає параметр "*pointsToWin",* який містить значення точок, виграних у кожному збільшенні, до "*scorePoints"* параметру, який містить значення поточної оцінки. Після обох активацій сценарій "*StoreOnScreen"* активується за допомогою введення "*оновлення"*.

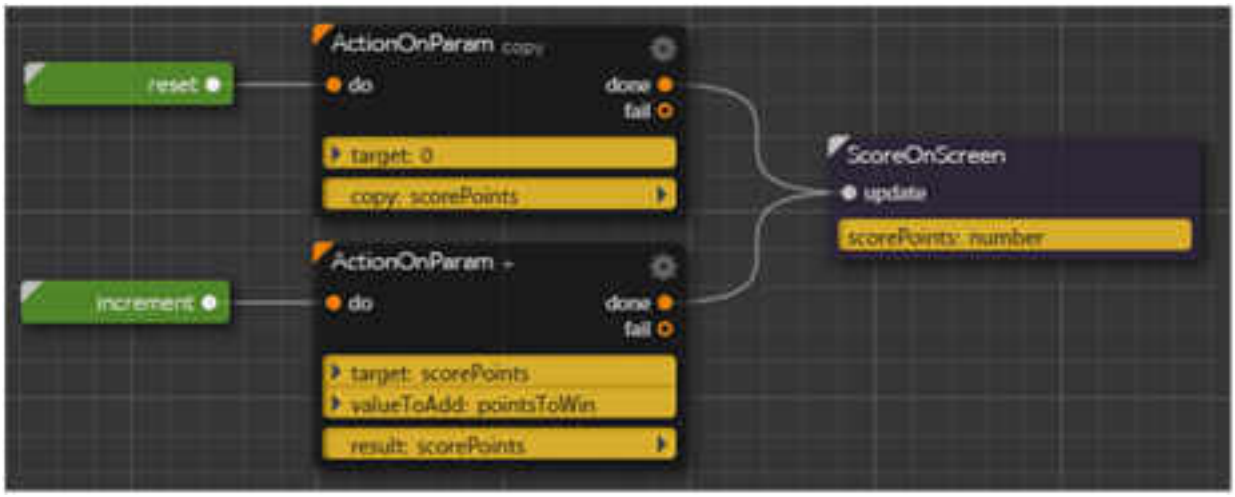

Рисунок 116. Сценарій підрахунку балів

Сценарій "*StoreOnScreen"* (рис. 117) має роз'єм для введення "*оновлення"* та розділяє параметр "*scorePoints"*, який містить значення поточного балу.

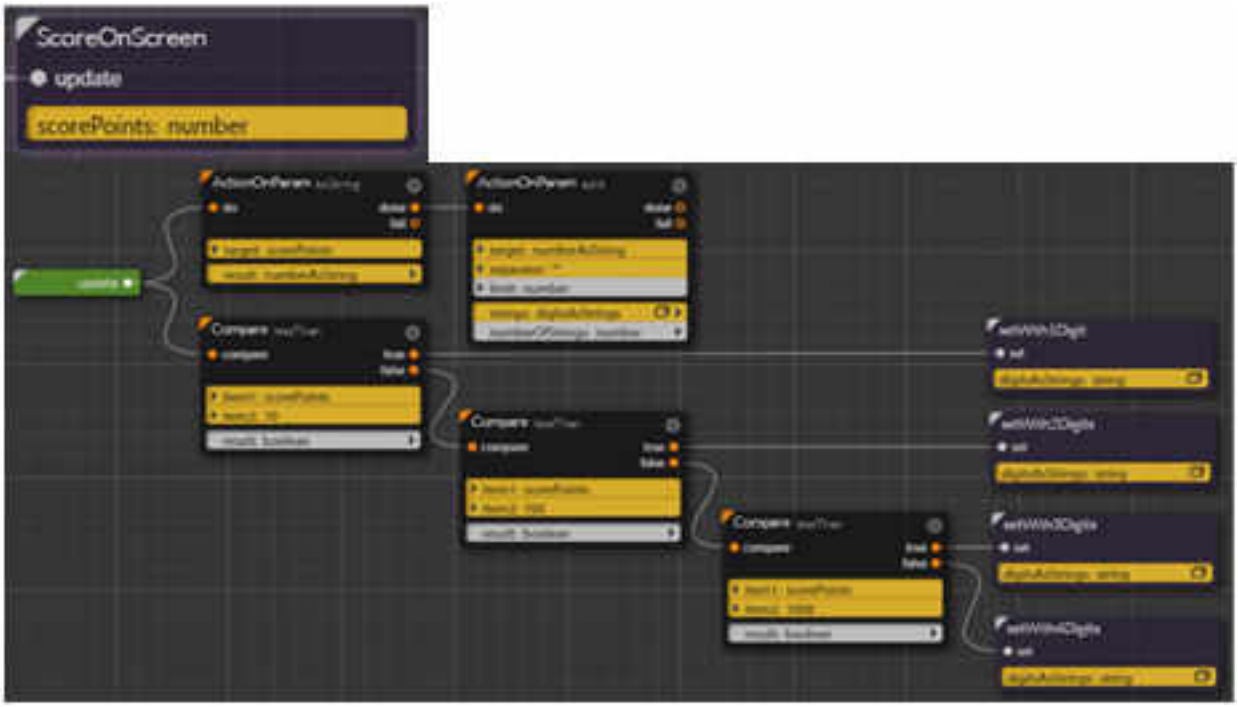

Рисунок 117. Сценарій оновлення значення

Після того, *як* сценарій «*ScoreOnScreen»* активований за допомогою введення «*оновлення»*, він починає перетворювати значення балів, що містяться в параметрі «*scorePoints»,* в текст через Blackbox *BlackBox ActionOnParam* до *toString*. Цей текст збирається в параметрі " *numberAsString"*. Після того, як текст буде отримано, ми розділимо його на

символи та *об'єднаємо* їх у колекцію Strings за допомогою *Split ActionOnParam.*

Ця колекція символів збирається в параметрі "*digitsAsStrings"*. Пізніше, на основі значення показника, який буде представлено, буде встановлено одну анімацію чи іншу для 4 Sprites. Якщо значення балу менше 10, як перевіряється " *lessThan" Compare* blackbox проти значення 10, його "*true"* виводу активується, що активує сценарій "*setWith1Digit"* (рис. 118).

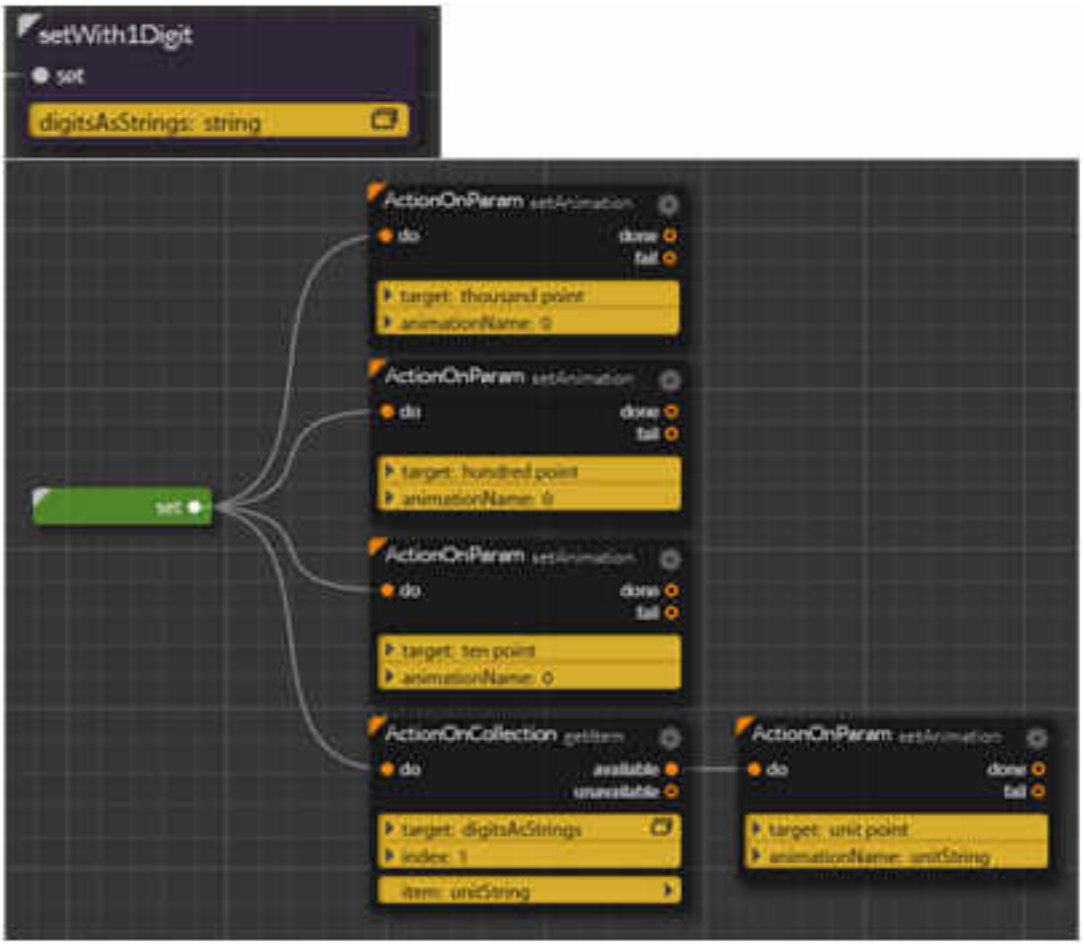

Рисунок 118. Сценарій оновлення першого розряду цифр

Якщо значення більше 10, активізується " *неправильний"* вивід *чорного ящика* , і він перевіряє, чи значення менше 100. Якщо *порівняння* blackbox перевіряє, що значення менше 100, його " *true"* вихід, який у свою чергу активує сценарій " *setWith2Digits"*.

Якщо значення більше 100, активується " *помилковий"* вивід *чорної скриньки* , і він продовжує перевіряти, чи значення менше 1000, що перевіряється " *lessThan" Compare* blackbox проти значення 1000. Якщо це Blackbox активує свій " *справжній"*висновок, після чого активує сценарій " *setWith3Digits"* . Якщо BLACKBOX активує « *помилковий* » висновок, то « *setWith4Digits»* Скрипт активується.

Сценарій " *setWith1Digit* " приймає перший символ з колекції символів і використовує його для встановлення анімації Sprite, яка відповідає місцю одиниць, що містяться в параметрі " *unit.point"* . Решта спрайт (« *десять . Пункт* », « *сто . Пункт* " і " *тисячі . Точка* ») встановлюється з «0» анімацією.

" *SetWith2Digits* " (рис. 119) бере першу з колекції символів і використовує її для встановлення анімації Sprite, яка відповідає *десятим місцям, що* містяться в параметрі " *ten.point"* , а другий символ колекції встановлюється з анімацією Sprite, що відповідає місцю одиниці, що міститься в параметрі " *units.point"* . Решта Sprites (" *hundred.point"*) і (" *thousand.point"*) встановлюються з анімацією "0".

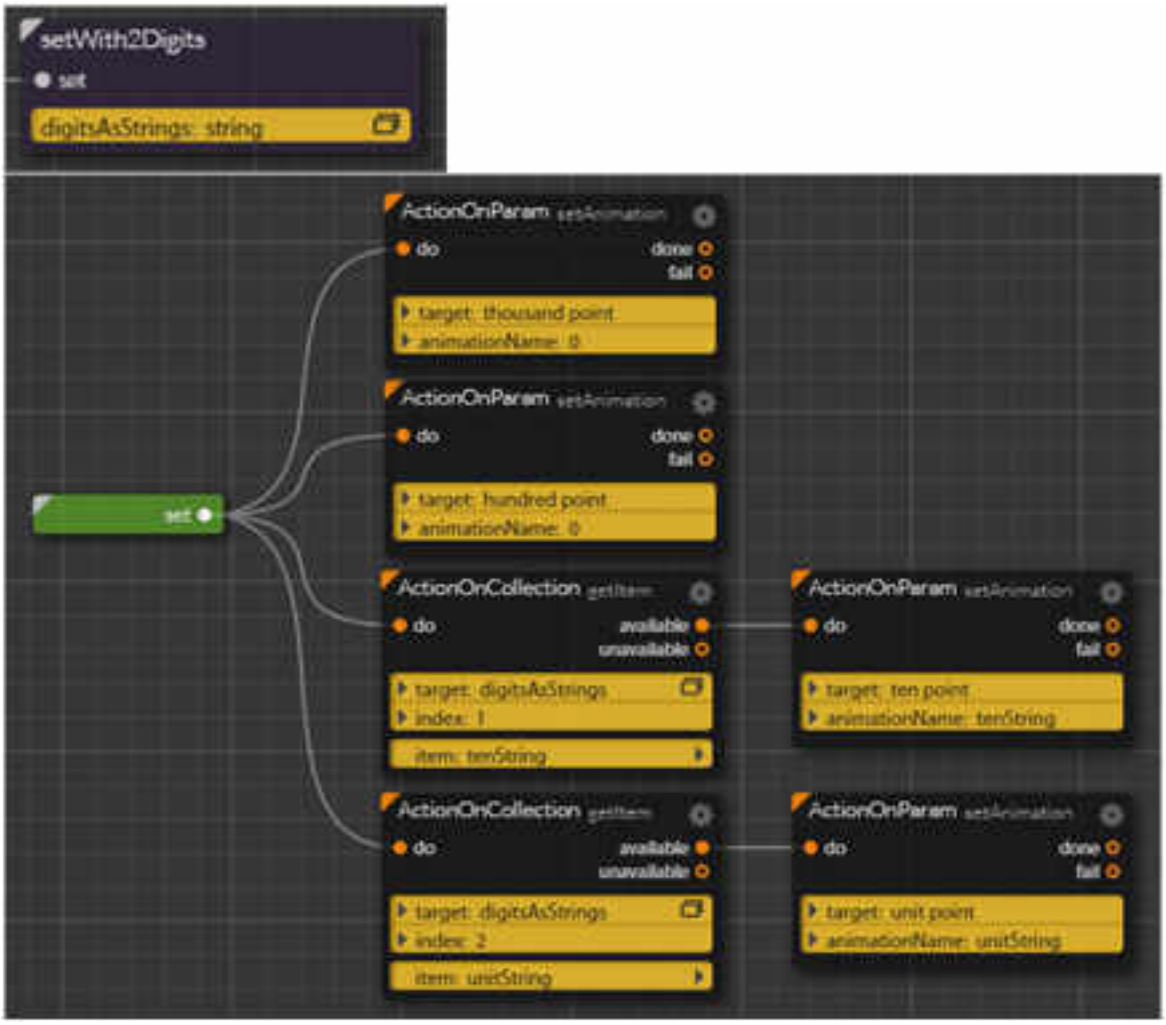

Рисунок 119. Сценарій оновлення другого розряду цифр

" *SetWith3Digits* " (рис. 120) бере першу з колекції символів і використовує її для встановлення анімації Sprite, яка відповідає сотням місць, що містяться в " *hundred.point"*); другий символ у збірці встановлюється з анімацією для Sprite, що відповідає *десятим місцям, що* містяться в параметрі " *ten.point"* ; а третій символ у збірці встановлюється з анімацією для Sprite, що

відповідає місцю одиниць, що містяться в параметрі " *unit.point"* . Решта Sprite, (" *thousand.point"*) встановлюється з анімацією "0".

Сценарій " *setWith4Digits"* (рис. 121) приймає перший символ колекції символів і використовує його для встановлення анімації Sprite, що відповідає тисячам місць, що містяться в параметрі " *thousand.point"* ; другий встановлюється з анімацією для Sprite, що відповідає сотням місць, що містяться в параметрі " *hundred.point"* ; третій встановлений з анімацією для Sprite, що відповідає десятковій місцевості, що міститься в параметрі " *ten.point"* ; а четвертий встановлюється з анімацією для Sprite, що відповідає місцю одиниць, що містяться в параметрі " *unit.point"* .

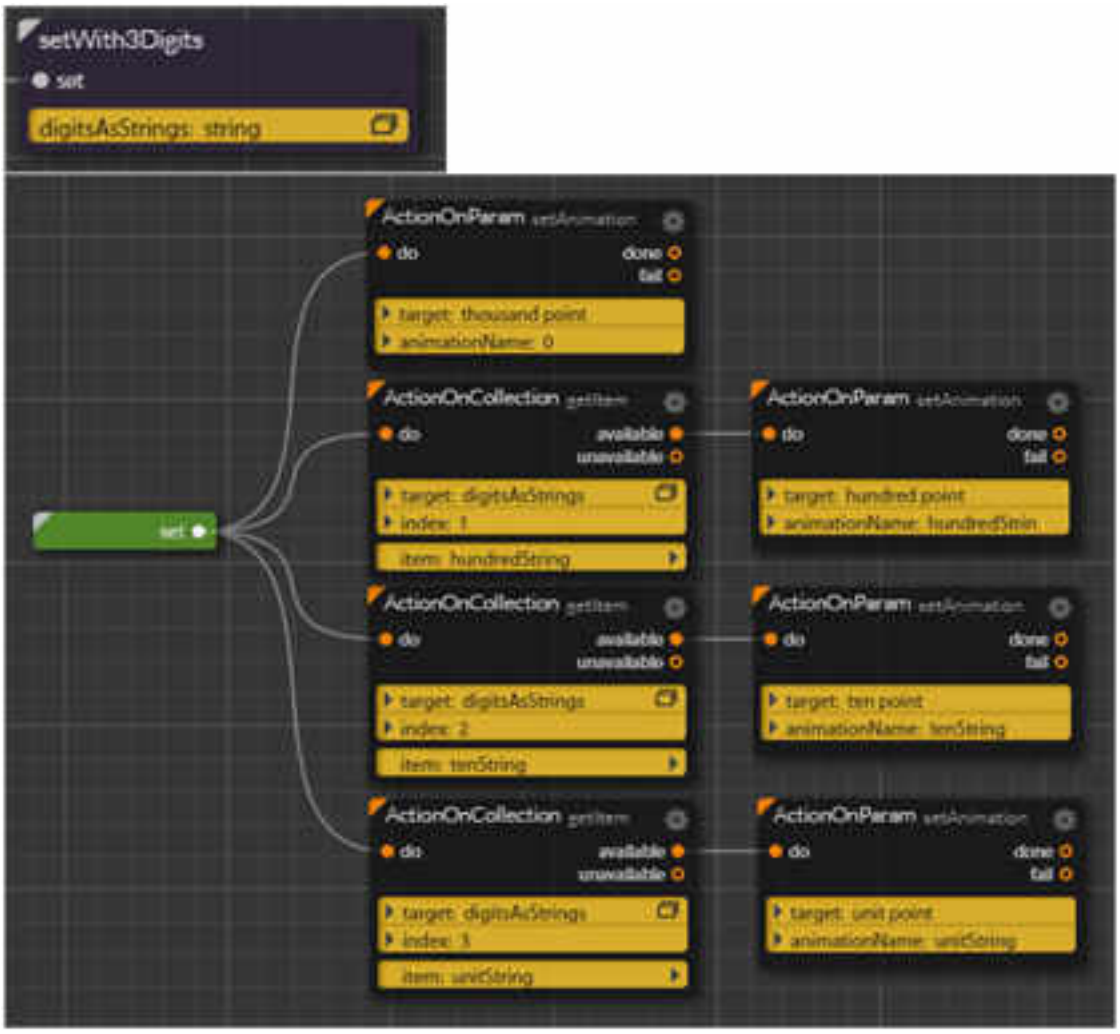

Рисунок 120. Сценарій оновлення третього розряду цифр

Рисунок 121. Сценарій оновлення четвертого розряду цифр

ЗБЕРІГАННЯ СЕАНСІВ ГРИ

Необхідно пояснити, як зберегти дані матчу за допомогою *пам'яті сеансу гри* . Перше, що потрібно зробити, - це те, що ця функція призначена для збереження стану матчу в певному **контексті** . І що ми маємо на увазі за контекстом? Найкращий спосіб пояснити це на прикладі.

Уявіть собі можливість створити турніри для певної гри. Таким чином, гравцеві для однієї гри можуть бути декілька матчів турніру з одним набором характеристик, іншими матчами, відведеними в іншому турнірі з іншими характеристиками, і навіть звичайні матчі поза турнірами. Всі ці ситуації (різні турніри, звичайні матчі) - це те, що ми називаємо контекстами.

Тобто, можливо, що один гравець в одній грі зберігає кілька різних даних матчів залежно від контексту в момент збереження. Продовжуючи наш приклад, в грі буде серія конкретних даних для турніру, різні дані для іншого турніру тощо. Ця інформація належить платформі і недоступна в ході програмування гри.

ЯК ЗБЕРЕГТИ ДАНІ МАТЧУ?

Перш ніж почати зберігати дані, потрібно приймати деякі рішення, виходячи з характеру гри та збережених даних. *Зберігання сеансів ігор* дозволяє нам зберігати дані локально або на сервері (сервер доступний лише для гравців, зареєстрованих на платформі), тому перше, що ми повинні вирішити, - які дані ми повинні зберігати та де.

Ви можете прийняти це рішення на основі наступних критеріїв:

- Вони повинні зберігатися на сервері, коли:
  - Ви хочете, щоб програвач міг продовжувати однаковий матч на кількох різних пристроях.
  - Дані важливі для цілісності матчу.
- Вони повинні бути збережені локально, коли:
  - Ви хочете, щоб програвач міг грати без активного з'єднання.
  - Дані, які потрібно зберегти, не є критичними.

Як правило, краще зберігати все, що ви можете на місці, крім найбільш чутливих даних для матчу (рахунок, життя, рівень, гроші тощо).

ПАНЕЛЬ SESSION STORAGE

Проаналізувавши потреби в іграх, прийшов час до роботи. Щоб почати, *редактор ігор* надає плагін, щоб мати змогу визначати збережені дані та де їх зберігати.

Щоб отримати доступ до цього плагіна, виберіть його на будь-якій відкритій панелі в *редакторі логіки* (рис. 122).

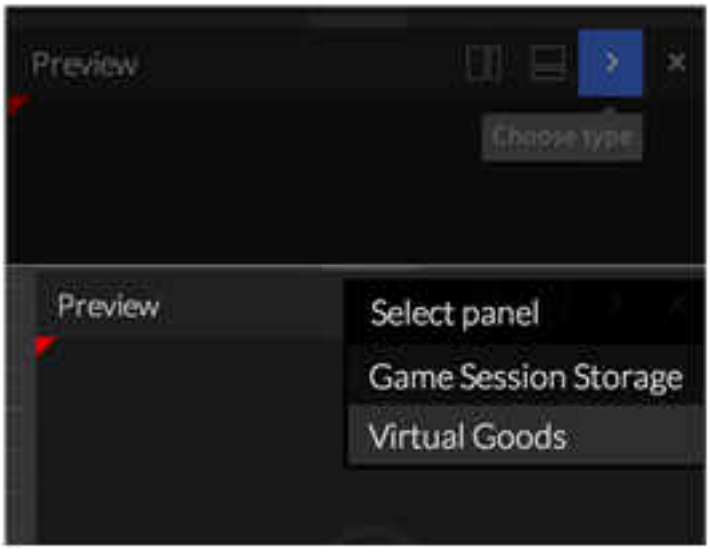

Рисунок 122. Плагін для збереження даних

При цьому відкриється наступна панель (рис. 123).

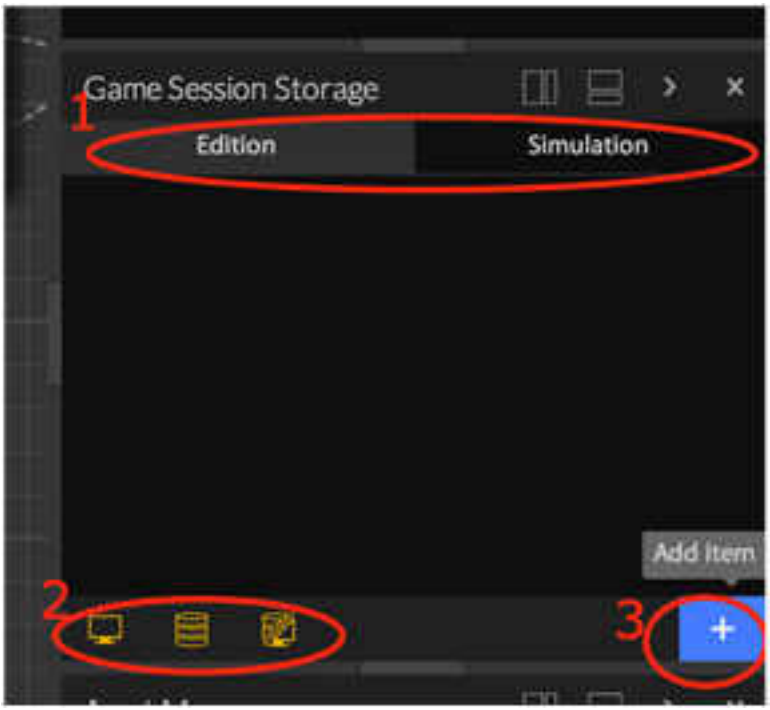

Рисунок 123. Панель збереження даних

По-перше, у (1) видно, що плагін має 2 вкладки:

- *Випуск*. Вкладка Видання дозволяє нам визначати дані, які ми хочемо зберегти, і які їх значення за замовчуванням.
- *Моделювання*. На вкладці "Моделювання" ви можете бачити в реальному часі значення, які визначають дані на вкладці "Видання", і ми також можемо їх редагувати в будь-який час під час виконання гри; тобто коли ми натискаємо кнопку відтворення у вікні *попереднього перегляду*

Коли ми створили багато ключів збереження, ми можемо відфільтрувати список за допомогою кнопок фільтрації (2). Зліва направо кожен значок означає "локально", "на сервері" і "локально / на сервері" (докладніше про це пізніше).

Щоб визначити дані, які ми збираємося зберегти, ми створимо те, що ми називаємо "збереження ключів". Щоб створити одну з цих клавіш, використовуйте кнопку "+" (3).

Натиснувши на нього, з'явиться наступне діалогове вікно (рис. 124).

Як ви можете бачити на зображенні, для ключа збереження потрібно визначити 4 властивості:

1. *Поле зберігання*. Це властивість визначає, де будуть зберігатися ігрові дані, і має 3 можливі значення: локально, на сервері або локально / на сервері. **Дуже важливо зрозуміти відмінності між цими 3-ма значеннями, оскільки в залежності від того, як ми хочемо використовувати службу збереження, нам доведеться скористатися одним чи одним варіантом**.

Рисунок 124. Діалогове вікно

1. Локально Встановлення локального ключа означає, що значення буде збережено на машині, яка запускає гру. Ця перевага полягає в тому, що з'єднання не потрібне для того, щоб мати можливість отримати або зберегти значення. Недоліком є те, що це значення не може бути доступним для інших пристроїв, і що для нього нема такого типу контролю безпеки, що означає, що його можна змінити назовні. Основне призначення - зберігання некритичних даних або збереження даних, які ми хочемо бути доступними без з'єднання.
2. На сервері. Ключі, збережені на сервері**, доступні лише для гравців, які зареєстровані на платформі** , і доступні з будь-якого пристрою. Якщо анонімний плеєр будь-яким чином намагається отримати доступ до ключа сервера, він отримає повідомлення про помилку доступу. Використовуйте цей тип, якщо хочете зберегти дані критичного матчу для певного гравця (рівень, рівень життя тощо).
3. Локально / на сервері. Встановивши цей параметр, **ключ буде збережений на сервері для зареєстрованих гравців та локально для незареєстрованих гравців**. Загальноприйнятою помилкою є думка, що ключ зберігається на сервері, якщо є з'єднання, і локально, якщо його немає, і **це неправильно**. У випадку зареєстрованого користувача це діє так само, як "на сервері". Це дозволяє нам зберігати дані збігів, незалежно від того, чи зареєстрований програвач.

2. *Тип*. У цей час ви можете зберегти 3 типи даних: булеві (true / false), числові та рядки.
3. *Ім'я*. Назва, яка ідентифікує ключ, який повинен бути унікальним. Нам не дозволено зберігати дві ключі з однаковим ім'ям.
4. *Значення за замовчуванням*. Це ключ за замовчуванням. Якщо ми не збираємося зберігати ключ (локально або на сервері), і ми просимо про це, ми отримаємо це значення.

Після того, як ви встановите властивості, ви можете натиснути кнопку OK, щоб закінчити створення ключа.

Після того як всі ключі, необхідні для збереження стану гри, створені, ви повинні мати щось схоже на зображення нижче (рис. 125).

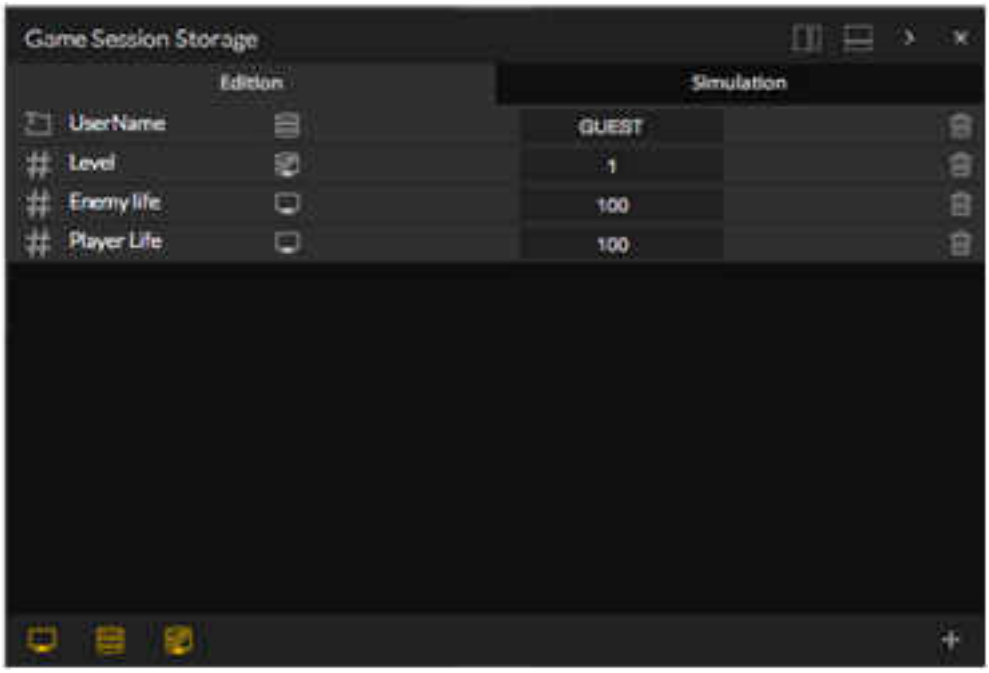

Рисунок 125. Результат створення ключів

Для кожного ключа ви можете змінювати лише середовище (натискання піктограми) та значення за замовчуванням. Якщо вам потрібно змінити назву або назву, вам доведеться видалити ключ і знову створити його.

ПРИКЛАД

На приладовій панелі є клонований проект під назвою *космічний* боєць. Ця гра зберігає поточний рівень матчу, ім'я гравця (для зареєстрованого користувача) і, якщо рівень відмовляється на півдорозі, життя гравця та ворога. Параметри, обрані для створення ключів, були:

- *UserName*. Збережено на сервері, оскільки для анонімних гравців ми будемо використовувати лише ім'я "гість". Це тип рядка, і значення за замовчуванням - "ГІСТЬ"

- *Рівень*. У випадку входу в систему ми будемо зберігати рівень на сервері (таким чином, що відповідність може бути продовжено на будь-якому пристрої) і локально, якщо користувач не ввійшов (локально / на сервері). Це числовий тип, а значення за замовчуванням - "1".
- *Вражаюче життя* та *життя гравця*. Ці ключі не є важливими, тому ми завжди будемо зберігати їх локально. Якщо ключі існують, ми продовжуватимемо матч із значенням, яке вони мають, а якщо ні, то ми будемо використовувати значення за замовчуванням, щоб визначити життя гравця та противника (100).

ЯК ЗБЕРЕГТИ ДАНІ МАТЧУ?

Для роботи з цими даними було створено 3 спеціальних *чорних ящиків* (вкладка *GameSessionStorage*), які працюють однаково:

- *GetGameSessionData*. Дозволяє нам прочитати дані, збережені у вибраних клавішах. Якщо ні в якому з ключів нічого не збережено, значення за замовчуванням для цього ключа буде повернуто.
- *SetGameSessionData*. Це використовується для збереження значень вибраних клавіш.
- *ResetGameSessionData*. Встановлює значення за замовчуванням для вибраних клавіш.

Якщо ви перетягнете один із цих чорних ящиків у сценарій та виберете його, у його властивостях з'явиться таке (рис. 126).

Рисунок 126. Набір чорних ящиків

1. *Безпечне* властивість є загальним для інших blackboxes, які виконують дії на сервері (наприклад, віртуальні товари). Якщо це активовано (правда), і *чорна скринька* буде запущена, якщо буде зроблена спроба її знову запустити, і не було відповіді від попередньої петиції, *виклик onPrevious* буде запускатися і ця петиція буде опущена. Якщо це буде деактивовано, всі клопотання, які досягають його, будуть надіслані.
2. *Усі* і *ніхто* просто не означає, що потрібно вибрати всі або ключі (3).
3. Ось ключі, які ми визначили для нашої гри, перераховані, а які ті, які ми хочемо читати, зберігати чи перезапустити, визначаються шляхом визначення значення "true".

У цих чорних ящиків є 4 можливі виходи:

- *на попередньому*, вже пояснено
- *доступний* або *виконаний* (залежно від Blackbox) спрацьовує, коли операція виконана правильно.
- *помилка*. Запускається, коли в операції виявлено будь-який тип помилки.
- *Acces denied*. Цей випуск спрацьовує, коли гра намагається отримати доступ до ключа, який зберігається на сервері, а програвач є анонімним (не ввійшов у платформу).

З усього, що ми вже охопили, у вас є все, що потрібно для того, щоб зберегти стан матчу для кожного гравця.

Повертаючись до прикладу *польоту* , ми маємо в основному 3 даних, які нам потрібно отримати:

1. Ім'я гравця.
2. Поточний рівень.
3. Життя залишилося для ворога та гравця.

Перше, що вам потрібно мати на увазі, полягає в тому, що гра призначена для того, щоб грати як анонімно, так і як зареєстрований користувач. Якщо гравець зареєстрований, він матиме певні додаткові функції, які в цьому випадку запитують його ім'я під час створення нового збігу та збереження цього імені на сервері, щоб привітати його до кожного матчу. Перше, що ми подумали, було, починаючи з гри, поставити *GetGameSessionData* і отримати три дані; проблема в тому , що якщо гравець не зареєстрований, він завжди отримати *AccessDenied* вихід, так як незареєстрований гравець , не може отримати доступ до даних сервера. Отже, нам потрібно одержати дані сервера в один бік, а дані локальної та локальної / серверної - іншим.

Перше, що ми будемо робити, коли починається гра - це спробувати отримати назву гравця (меню «Скрипти», щоб отримати назву гравця), використовуючи *чорнову ящик GetGameSessionData* і налаштувавши його, щоб отримати лише ключ «UserName». Якщо *доступний* вихідний сигнал спрацьовує, це означає, що програвач зареєстрований, і він дасть мені збережену базу даних або значення за замовчуванням, доки нічого не буде

збережено. Якщо *виклик accessDenied* спрацьовує, це означатиме, що програвач анонімний, тому ми не будемо просити назви гравця (рис. 127).

Рисунок 127. Анонімізація гравця

Щоб зберегти ім'я гравця в грі, натиснувши кнопку «*Почати нову гру*», якщо програвач не анонімний, йому буде запропоновано ввести назву гравця, а при натисканні кнопки «*Повернення*» ім'я буде збережено за допомогою параметра *SetGameSessionData* blackbox (меню скриптів -> введіть ім'я).

Якщо він натискає текст "*Продовжити останню гру*", ми отримаємо збережені дані, скориставшись *GetGameSessionData,* і встановимо *чорний ящик* із усіма ключами, крім *UserName.* Таким чином, ми отримаємо життєвий геймер і ворог на місцевому рівні, а рівень від сервера повинен бути ввімкнутим гравцем або всім локально, якщо він анонімний. Як видно, поведінка інша, якщо ключ знаходиться на *сервері,* оскільки він завжди дає відповідь *доступний* (якщо не існує помилки).

Нарешті, дані оновлюються, коли рівень змінюється (скрипт *SetNextLevel*, у цьому випадку рівень оновлюється за допомогою *SetGameSessionData,* а *BlackBox ResetGameSessionData* використовується для встановлення живого значення від гравця та противника до значення за замовчуванням), а також коли Натискання кнопки "*Home"* під час відтворення (сценарій play à fight à home button, в цьому випадку зберігається лише життя).

Тепер нам достатньо завантажити та зберегти дані про гру для *космічного польоту*.

КЕРУВАННЯ ЗБЕРЕЖЕНИМИ ІГРАМИ

Для цього є вкладка **моделювання** (рис. 128).

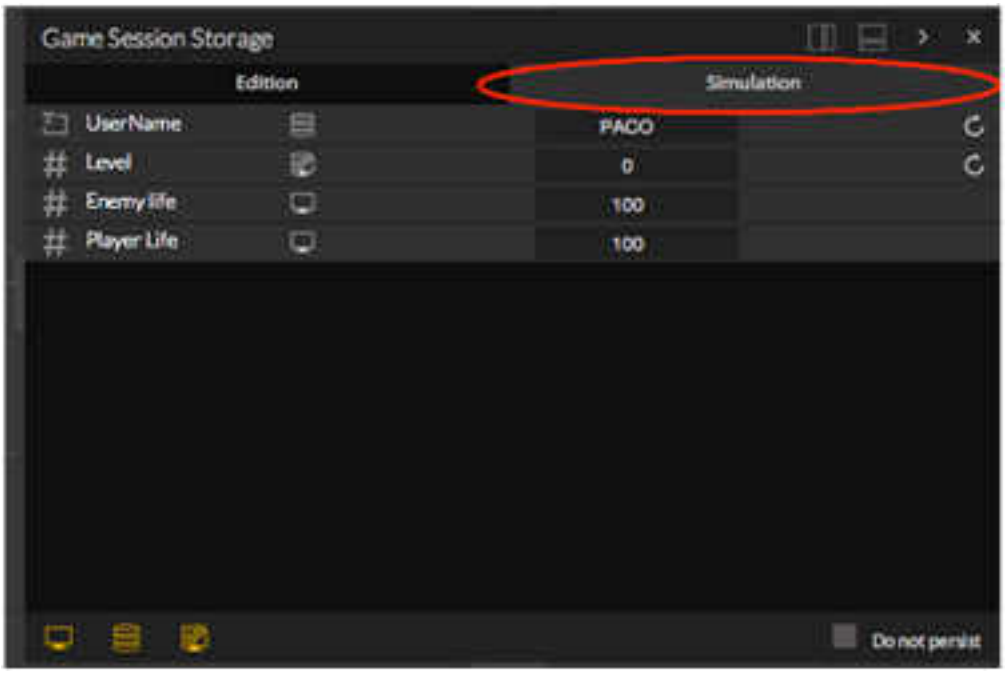

Рисунок 128. Вкладка Моделювання

На цій вкладці ми можемо побачити в реальному часі значення клавіш у редакторі. Є кілька варіантів (рис. 129).

Рисунок 129. Варіанти призначення клавіш у редакторі

1. Значення симуляції кожної клавіші. Це значення кожної клавіші; його вміст можна редагувати, і це дозволяє нам поставити гру в певний стан. Найбільш очевидним є його використання для встановлення умов запуску для матчу, але також може змінювати їх під час матчу.