Dokumentacja gry 3D na platformę Windows, zbudowanej za pomocą silnika do tworzenia gier Unity – wykorzystującej szyfr AES w rozgrywce

Maciej Przybyłowski

Infotmatyka III rok – studia stacjonarne

Bezpieczeństwo systemótw komputerowych

Szyfr AES (Advanced Encryption Standard)

AES – symetryczny szyfr blokowy przyjęty przez NIST jako standard FIPS-197 w wyniku konkursu ogłoszonego w roku 1997, który wygrał algorytm nazywany roboczo Rijndael.

Bezpośrednią przyczyną rozpisania konkursu była niewystarczająca siła algorytmu DES. W roku 1997 organizacja EFF była w stanie złamać wiadomość zaszyfrowaną DES-em w ciągu 3 dni sprzętem o wartości 250 tysięcy dolarów; obecnie można złamać DES-a jeszcze szybciej i taniej.

Do finału konkursu zakwalifikowało się pięć algorytmów szyfrujących (Rijndael, RC6, Mars, Serpent oraz Twofish), ze szczególnym wskazaniem na algorytm Rijndael. Możliwe jest w nim użycie kluczy o długościach 128, 192 i 256 bitów i operuje on na blokach danych o długości 128 bitów (oryginalna specyfikacja Rijndael dopuszczała również bloki 192- i 256-bitowe).

O algorytmie AES

AES wykonuje 10 (klucz 128 bitów), 12 (klucz 192 bity) lub 14 (klucz 256 bitów) rund szyfrujących. Składają się one z substytucji wstępnej, permutacji macierzowej (mieszanie wierszy, mieszanie kolumn) i modyfikacji za pomocą klucza. Funkcja substytucyjna ma bardzo oryginalną konstrukcję, która uodparnia ten algorytm na znane ataki kryptoanalizy różnicowej i liniowej.

Odmiany algorytmu Rijndael niebędące standardem AES, w zależności od długości klucza i bloku danych wykonują 12 lub 14 rund szyfrujących.

Opis działania programu aMaze

Aplikacja stworzona przeze mnie ma jedno główne zadanie, zapewnić użytkownikowi parę minut rozrywki. Dodatkowo dzięki implementacji algorytmu AES, może posłużyć jako ciekawy sposób zaprezentowania studentom – nie tylko sposobu implementacji takowego kodu w języku C# – ale również przedstawić możliwości zaawansowanego silnika do tworzenia gier oraz aplikacji UNITY. Implementacja algorytmu AES wykorzystuje 128 bitowy klucz co zapewnia 10 rund szyfrujących.

W grze sterujemy bohaterem, który wyposażony w latarkę niefortunnie znalazł się w szczelnie zamkniętym labiryncie. Jedyną możliwością na ocalenie jego życia jest odnalezienie czterech kryształów ukrytych w zakamarkach labiryntu. Zawierają one zaszyfrowane wiadomości które należy zapamiętać, a następnie odszyfrować przy pomocy konsoli/komputera który również ukryty jest w labiryncie. Po odszyfrowaniu wiadomości pozostaje nam odnaleźć drzwi, które otworzą się po wprowadzeniu prawidłowego zdania – składającego się z czterech słów.

Parę słów o mechanice gry

Sterujemy postacią za pomocą klawiszy WSAD, które umożliwiają nam poruszanie się do przodu, do tyłu oraz chodzenie bokiem (w lewo oraz w prawo), dodatkowo jesteśmy wyposażeni w latarkę która oświetla nam drogę. Możemy rozglądać się po otoczeniu za pomocą myszy komputerowej. Wszelkie interakcje z otoczeniem inicjujemy przy użyciu klawisza E na naszej klawiaturze. Kamera jest usytuowana w głowie gracza – a więc korzystamy z perspektywy pierwszej osoby co zwiększa poziom interakcji z grą i pozwala bardziej skupić się na grze. Naszym celem jest znaleźć wyjście z labiryntu o wymiarach 31 na 31 kostek. Aby wykonać nasze zadanie musimy najpierw zebrać pierwszy kryształ pomocniczy – znajdujący się zawsze na początku mapy, a następnie – zgodnie z instrukcją wyświetlaną po znalezieniu pierwszego kryształu – pozostałe cztery kryształy skrywające zakodowane hasła.

Jak to działa?

Przy każdorazowym uruchomieniu aplikacji tworzony jest losowo generowany labirynt. Zajmuje on pole o powierzchni 31 x 31 jednostek. Wydrążone ścieżki są unikatowe, nie ma więc możliwości aby gracz kilkukrotnie poruszał się tą samą trasą aby osiągnąć końcowy sukces. Położenie kryształów oraz drzwi w labiryncie również jest losowe. Miejsce startowe naszej postaci jest statyczne, podobnie jak miejsce pojawiania się konsoli/komputera dekodującego.

Budowa aplikacji

Tworzenie aplikacji w środowisku UNITY opiera się na umieszczaniu w obszarze sceny trójwymiarowych obiektów oraz przypisywaniu do nich skryptów określających w jaki sposób mają się zachowywać w środowisku gry. Do wyboru istnieją trzy języki programowania: C#, Javascript oraz Boo. Wybór danego języka programowania do zaplanowania zachowania jednego z elementów gry (np. poruszania się bohatera), nie zmusza nas do ograniczania się tylko do tego języka w procesie tworzenia aplikacji. Dlatego cześć skryptów może być napisana w języku C#, a część np. w języku Javascript. Co również wykorzystałem w mojej aplikacji:

Aplikacja składa się z licznych elementów graficznych przedstawiających:

- Ściany labiryntu,
- Ściany lokacji początkowej,
- Konsole/komputer,
- Klawiaturę do wprowadzania odszyfrowanego zdania,
- Drzwi wyjściowe z labiryntu
- Postać bohatera,
- Kryształy z zakodowanymi wiadomościami
- Podłoże,
- Niebo,

Dodatkowo w aplikacji występują efekty dźwiękowe takie jak:

- Dźwięk poruszania się
- Dźwięk otwierania drzwi
- Muzyka odtwarzana w tle

Aplikacja składa się z następujących skryptów

MazeGenerator.cs – odpowiedzialny za utworzenie labiryntu oraz wszystkich elementów znajdujących się w nim, (takich jak kryształy lub drzwi).

WalkSound.js – odpowiedzialny za obsługę dźwięków towarzyszących poruszaniu się.

Crys.cs – odpowiedzialny za obsługę zachowań kryształów (niszczenie znalezionych kryształów) oraz przetwarzanie informacji w nich przechowywanych.

MouseLook.cs – odpowiedzialny za możliwość rozglądania się za pomocą myszy komputerowej.

CharacterMotor.js – odpowiedzialny za motorykę naszego bohatera.

Spot.cs – odpowiedzialny za pracę latarki.

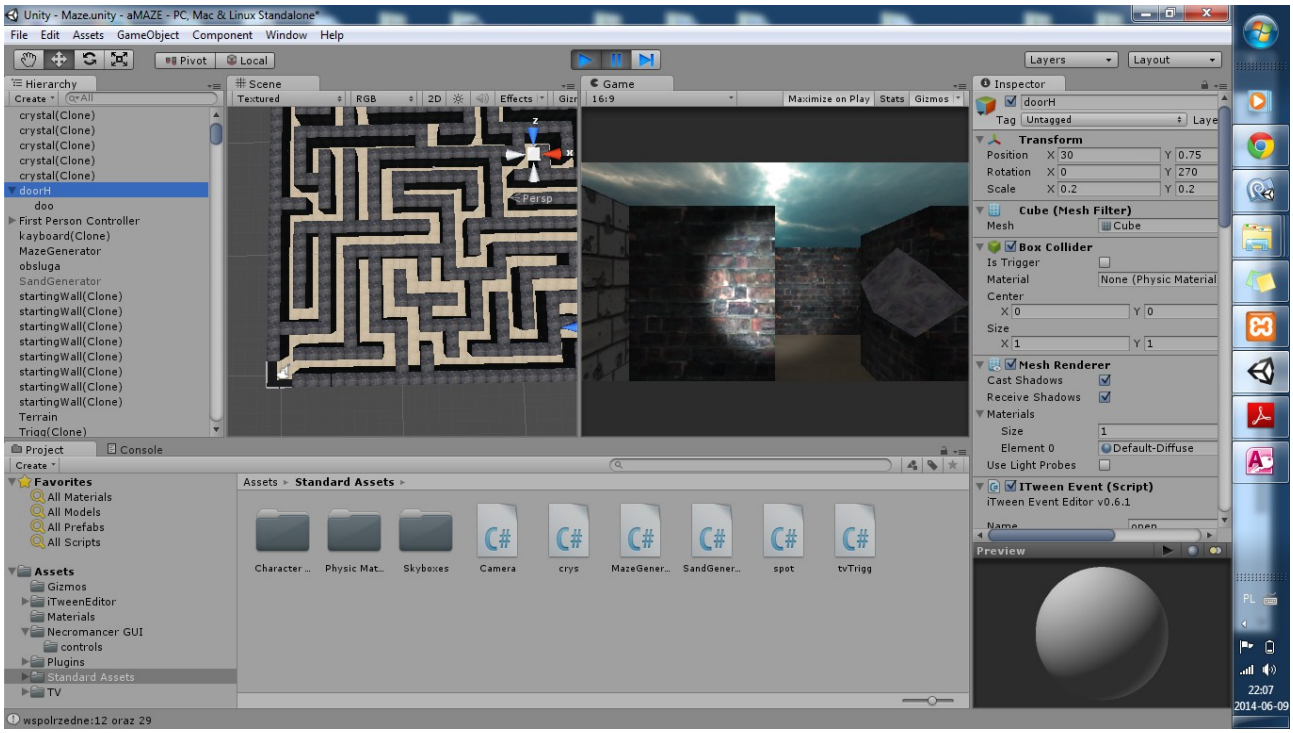
Camera.cs – odpowiedzialny za pracę kamery.

Trigger.js – odpoweidzialny za proces kolizji obiektów.

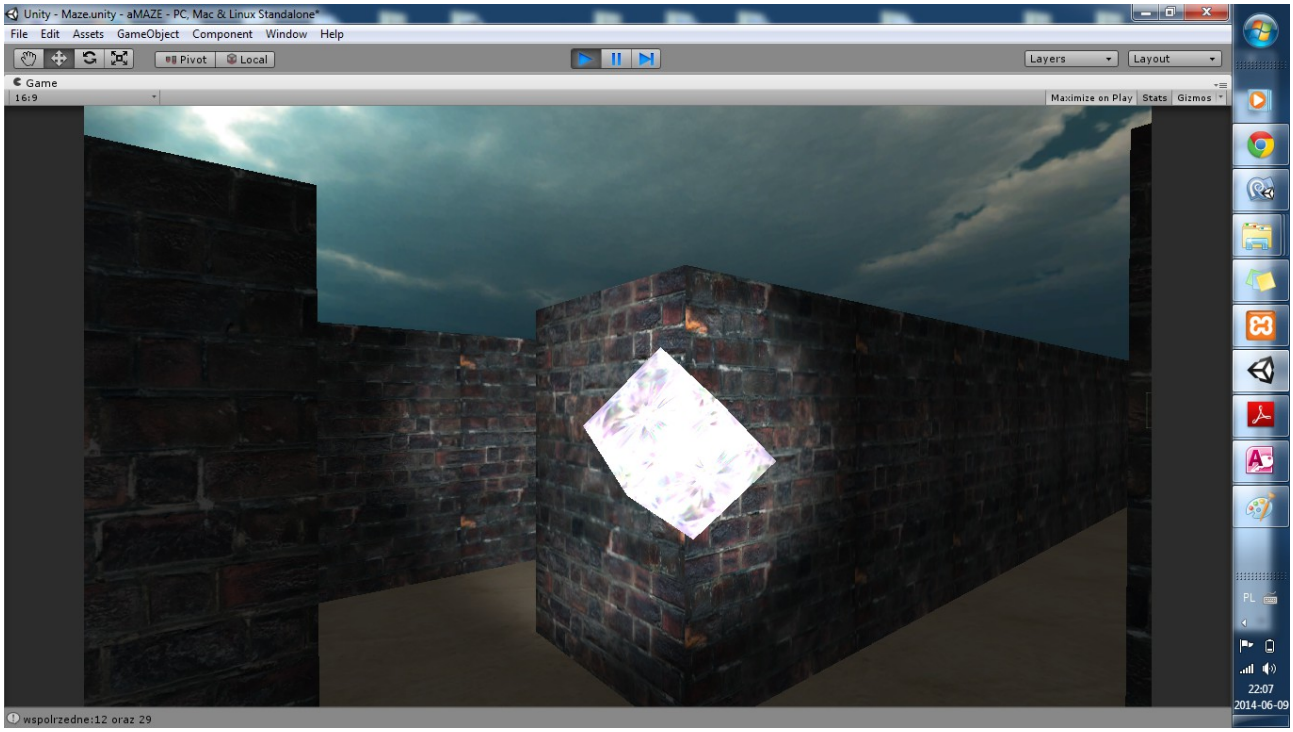
ItweenEvent.cs – odpowiedzialny za proces otwierania drzwi.

Zrzuty ekranu z aplikacji

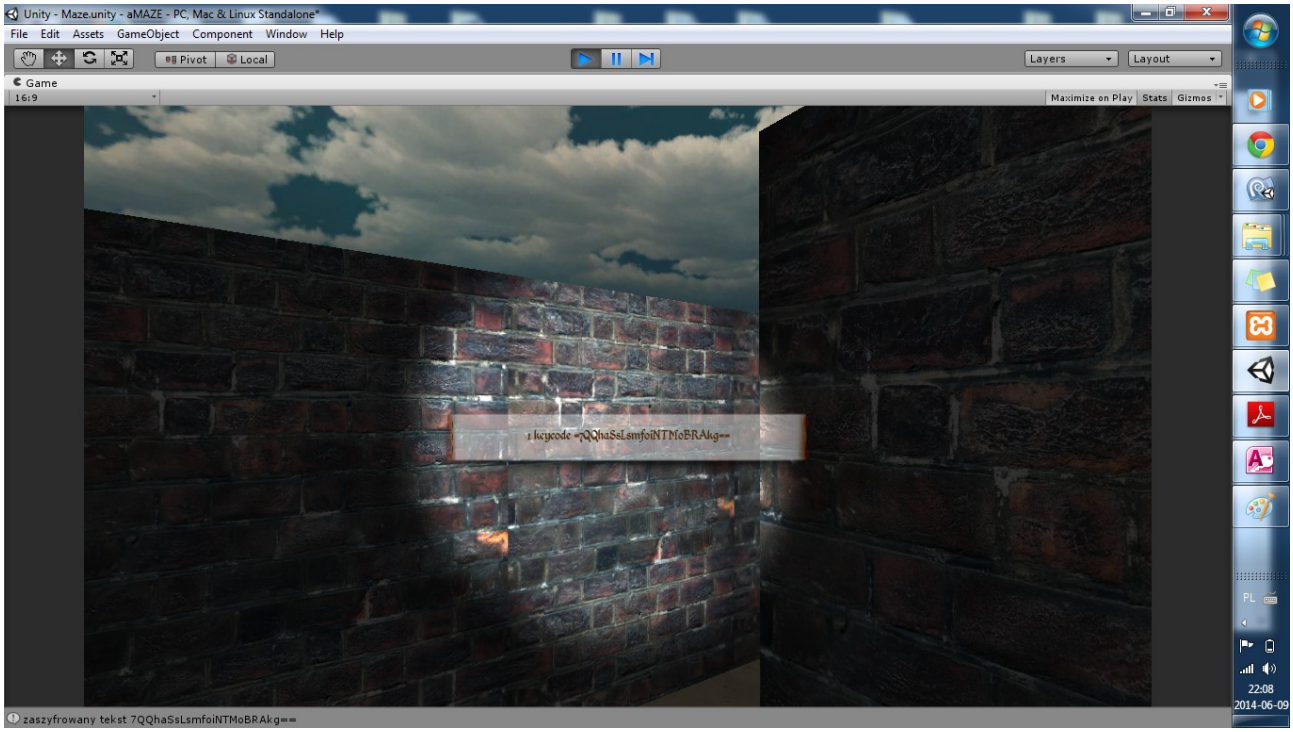
Ilustracja 1: Wczesna wersja aplikacji



Ilustracja 2: Widok projektu w środowisku Unity

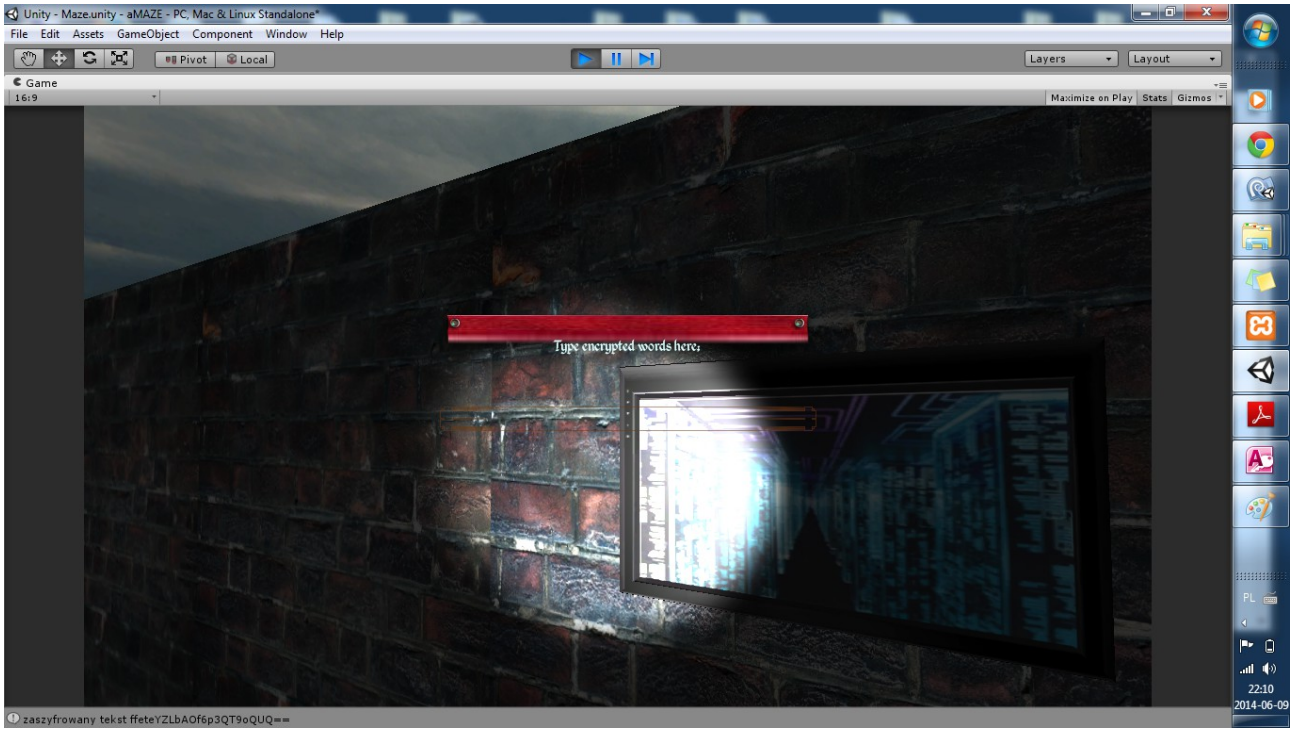


Ilustracja 3: Wygląd aplikacji bezpośrednio po uruchomieniu jej w emulatorze

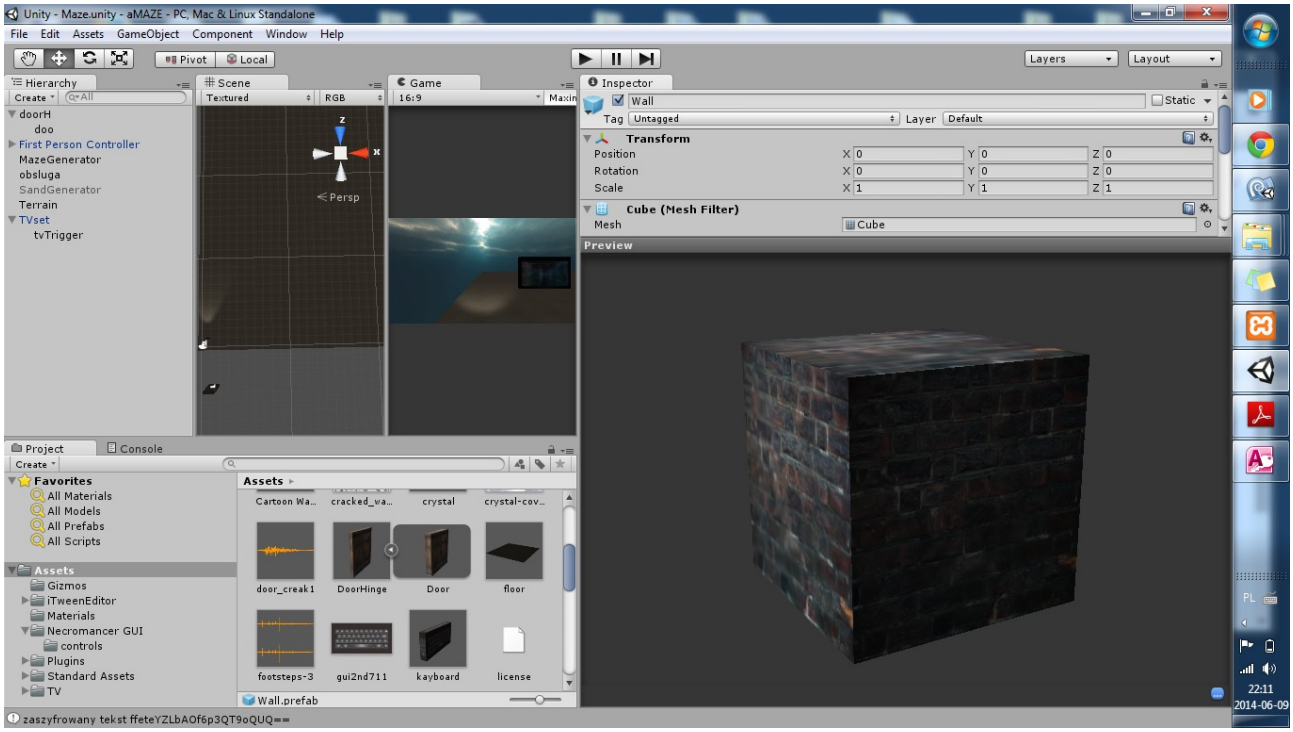


Ilustracja 4: Widok gry chwilę po zebraniu kryształu z zaszyfrowaną wiadomością

Ilustracja 5: Implementacja graficzna drzwi



Ilustracja 6: Ekran deszyfrujący



Ilustracja 7: Model ściany labiryntu

Możliwości rozbudowy aplikacji

Gry aMaze nie można traktować jako pełnoprawnego produktu. Jest to jedynie demo technologiczne i graficzne zaprezentowanie pewnego pomysłu na pełną wersję gry/ materiału dydaktyczne. Grę można rozbudować o dodatkowe poziomy. Oprócz dostępnego obecnie poziomu wykorzystującego algorytm AES, możnaby utworzyć poziomy łatwiejsze, wykorzystujące np. szyfr Cezara lub szyfry przestawieniowe. W przyszłych wersjach aplikacji z pewnością powinien pojawić się przyjemny dla oka interfejs użytkownika. Koniecznym dodatkiem wydaje się być utworzenie menu głównego aplikacji, w którym użytkownik będzie mógł zmienić ustawienia sterowania, zwiększyć głośność aplikacji lub też wybrać, który z poziomów aplikacji chciałby rozegrać. Równie istotnie zdaje się być dodanie przycisku pauzy (dostępnej pod przyciskiem escape), który umożliwi zatrzymanie gry. Równie istotne mogłoby być dodanie wirtualnego przeciwnika/potwora. Śledziłby on nasza postać po labiryncie i wymuszał szybkie podejmowanie decyzji co do kierunku poruszania się. Każdorazowe zebranie kryształu mogłoby skutkować przeteleportowaniem potwora w inne miejsce mapy.

Program można rozwinąć na wiele różnych sposobów, wymienilem zaledwie te kilka z nich które uznałem za najistotniejsze.