Dokumentacja gry PointerNString

By: Wiktor Kobryń

Spis treści

Wstęp

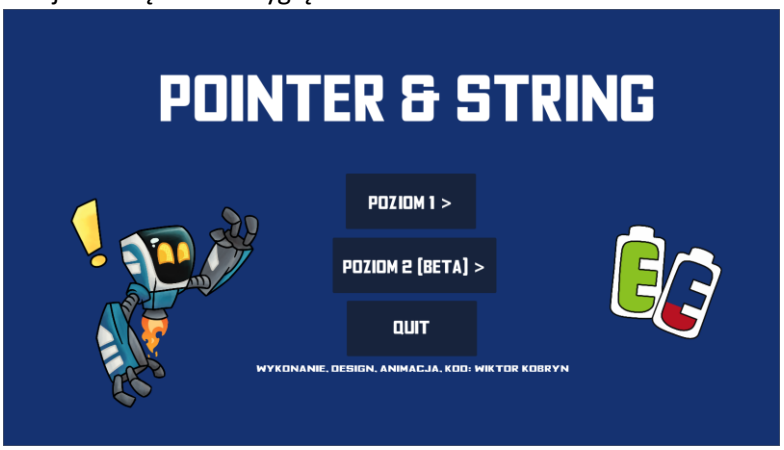
Gra powstała specjalnie na potrzeby tegorocznego (2023) ŚFN pod specjalny magiczny monitor z lupami bez folii polaryzacyjnej, specjalnie na tenże event i początkowo miała być napisana w Unity ale pojawiły się 2 kluczowe przeszkody:

a) Unity stało się strasznie zamknięte na głos społeczności i dużo ludzi (np. piszących addony) zaczęło się wycofywać a to do UnrealEngine5 albo właśnie Godota
b) Unity ma dużo swoich magicznych „prawd objawionych" w projekcie, waży to dużo wszystko, setup jest czasochłonny i sama zmiana wersji często corruptuje pliki gry (NIE WARTO!).

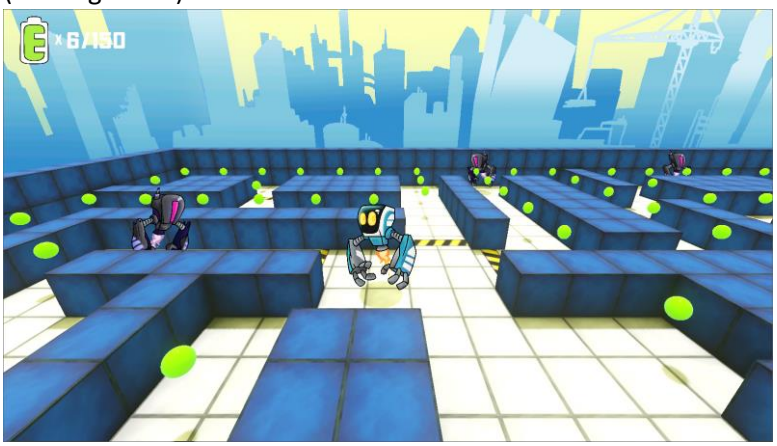
Tak więc uszkadzając kilka magicznych elementów w projekcie w Unity 2021 o których działaniu nie mam pojęcia stwierdziłem że ciekawsze będzie przepisanie całego szkieletu wtedy istniejącej gry na Godota (wsparcie C#, .NET, bardzo podobny w koncepcie i współdziałający z Blenderem – też na licencji GNU).

A jako że lubię robić sobie trudniej w życiu to zrobiłem z tego swojego rodzaju czterodniowy GameJam w którym powstała cała gra od grafik po animacje, modele i kod itp. przy okazji ucząc się środowiska i pracy w Godocie od zera (miałem minimalne doświadczenie z Unity).

Gra jak na tą chwile wygląda tak:



(menu główne)



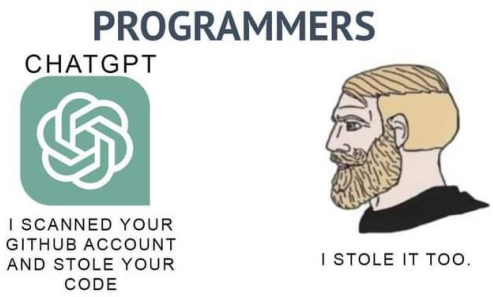
(real gameplay footage)

Także jak na tą chwilę szału nie ma ale działa wszystko.

Działanie monitora

Pozwoliłem sobie dodać tutaj trochę teorii jak to działa i dlaczego tak a nie inaczej widać wszystko na magicznym monitorze, sam to niezbyt rozumiem...

„Folia polaryzacyjna to cienka warstwa umieszczona na ekranie LCD (Liquid Crystal Display), która pomaga kontrolować przepływ światła i kierunek jego polaryzacji. Ta folia działa poprzez filtrowanie światła, które dociera do ekranu, i umożliwia wyświetlanie obrazu w sposób kontrolowany.
Ekran LCD składa się z dwóch warstw polaryzatorów, które są ustawione prostopadle do siebie. Gdy światło przechodzi przez pierwszy polaryzator, jego polaryzacja zostaje zmieniona. Następnie przechodzi przez warstwę kryształów ciekłych, które mogą kontrolować przepływ światła. Kiedy światło przechodzi przez drugą warstwę polaryzatora, to tylko światło, które ma odpowiednią polaryzację, jest przepuszczane, a reszta jest blokowana, co daje nam obraz na ekranie.
Okulary z polaryzacyjnymi soczewkami są zaprojektowane tak, aby dopuszczać tylko światło o określonej polaryzacji. Dzięki temu, gdy oglądasz ekran LCD za pomocą tych okularów, widzisz obraz, ponieważ dopuszcza on tylko światło o odpowiedniej polaryzacji. Jeśli chodzi o odległość, z jakiej widać efekt działania folii polaryzacyjnej, to zależy to od kąta, pod jakim patrzysz na ekran.
Folie polaryzacyjne na ekranach LCD czasami mają specyficzne właściwości, które sprawiają, że obraz jest widoczny tylko z pewnej odległości i pod odpowiednim kątem. Na przykład, gdy patrzysz na ekran z bliska, możesz zauważyć efekty polaryzacyjne związane z kątem pod jakim światło jest emitowane i filtrowane przez warstwy polaryzatorów, co powoduje, że obraz staje się trudniejszy do zobaczenia. Zwiększając odległość, kąt patrzenia na ekran zmienia się, co może spowodować, że efekt polaryzacyjny staje się mniej zauważalny lub zanika, co skutkuje lepszą widocznością obrazu. Jednakże, w przypadku różnych modeli ekranów LCD czy okularów polaryzacyjnych, te odległości mogą się różnić."

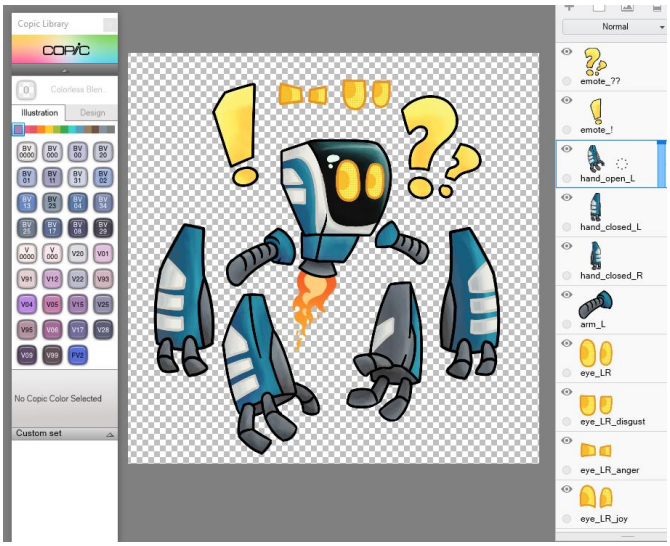


~by ChatGPT

Dobór oprogramowania

Autodesk Sketchbook

Sketchbook jest programem stricte nakierowanym do grafiki rastrowej, w nim zrobiłem szkice spritów, które potem zamieniłem na modularne „tekstury dla meshy" na odpowiednio nazwanych warstwach. Czemu tak? Żeby mieć mniej roboty bo mam skrypt do gimpa który konwertuje mi warstwy z pliku .psd / .tif na baze modelu do Spine z odpowiednim rozmieszczeniem elementów względem siebie.



(tak mniej więcej wyglądają takie części, oczywiście tutaj je porozrzucałem żeby pokazać)

Spine

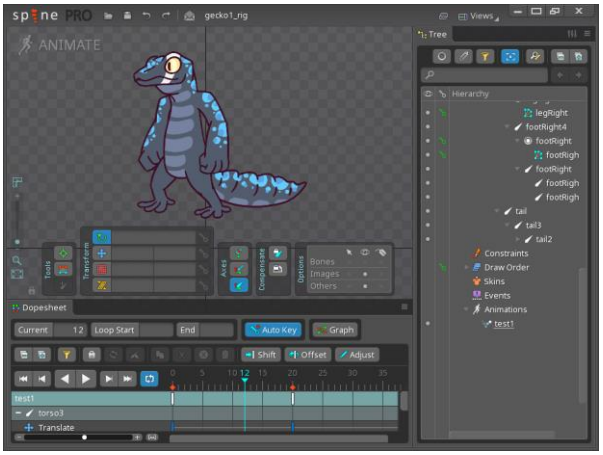
Tu zaczyna się zabawa bo spine to program/środowisko do animacji wektorowej 2D/2.5D, szczególnie nacechowany na tworzenie modeli z animacjami do gier i eksport takich właśnie modeli do Unity/Godota. Eksportowanie takie jest bardzo fajne ze względu na to ile mniej waży model od sekwencji png (na razie gra ma to 2 bo nie miałem czasu rozpracować jak to zrobić w Godocie ale da się i za pewne to o to rozwinę)

Same modele tworzy się i animuje bardzo podobnie do modeli 3D z małą różnicą bo sekwencja wygląda tak:

tekstury -> indywidualne meshe dla tekstur -> konstrukcja rig-a (animacja wektorowa oparta o bone/joint) -> przypisanie fragmentów meshy do odpowiednich kości -> animacja

Tutaj też ciekawostka bo użyłem też animacji klasycznej do ognia, też się da łączyć oba sposoby!

(Taki przykładowy mój model w spine trochę z innej beczki)



(i jakaś tam prosta animacja idle, tu niezbyt widać ale trzeba uwierzyć na słowo heh)

Polecam, lepiej wyjaśnione: https://esotericsoftware.com/spine-in-depth

Godot / tutorial dla znudzonych ŚFNem

Jeśli chodzi o samą grę to jak na tą chwilę trochę jest za bardzo klonem PACMANa ale z „spritami w świecie 3D”, nie taki był zamysł a jest to na razie takie demo technologiczne. Generalnie myślałem o czymś w stylu przenoszenia elementów z środowiska wokół gracza (czujniki ultradźwiękowe, kable, rury, koła, płytki arduino itp.) i montowanie z tego konkretnych robotów w celu rozwiązania łamigłówki – bardziej edukacyjnie niż arcade ale to kwestia przyszłego rozwoju.

Na razie lore jest taki że trzeba zbierać kule energii aby napełnić baterię, nie wpadając na pracujące roboty (bardzo skomplikowana fabuła).

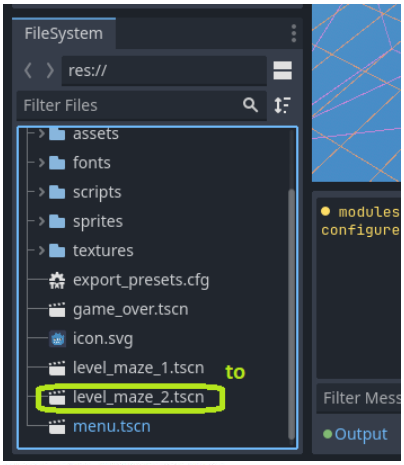
Setup:

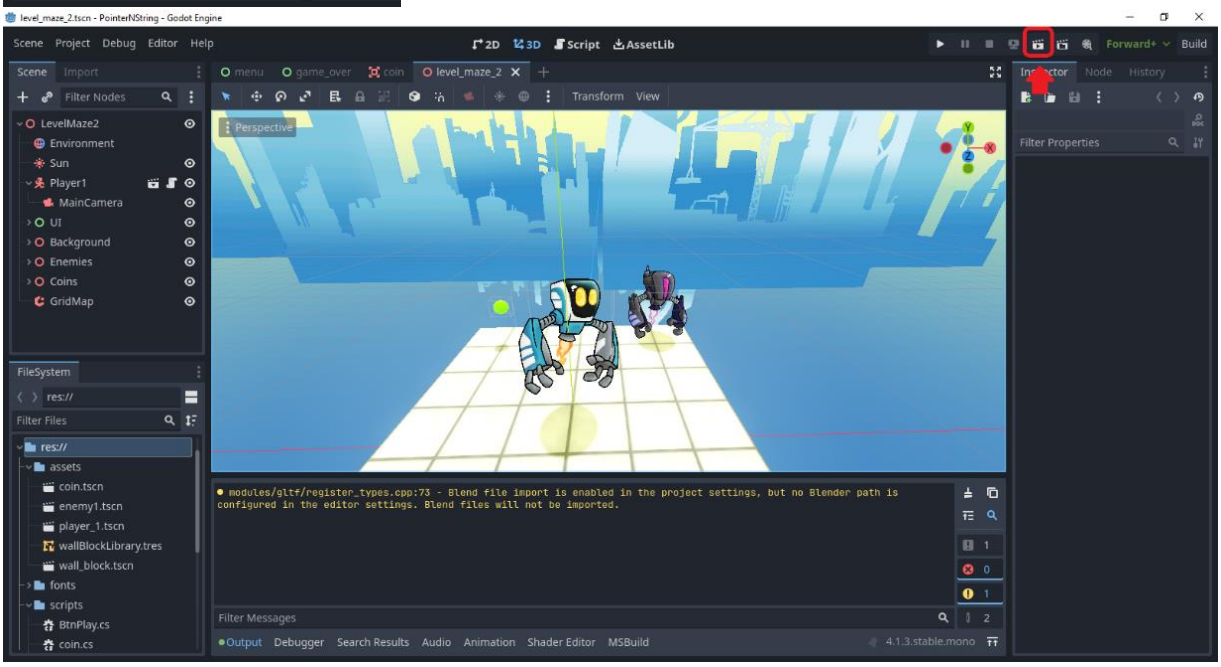
Tutaj nie będę już opisywał samego Godota i czarował czymś na czym się niezbyt znam a przejdę od razu do tworzenia dodatkowego poziomu. Jedyne co to chciałbym zauważyć że w Godocie można pisać skrypty w C#, C++ i GDscript (mix pythona i dziwnej składni)

Najpierw warto pobrać Godota (4.2 .NET) i zaimportować projekt pobrany z githuba/z pendriva jeśli się taki pojawi ode mnie :)))

Repo: https://github.com/wiktorkobryn/PointerNString.git

W menu głównym dodałem opcje wejścia do dodatkowego poziomu 2, na razie wygląda on tak (plik level_maze_2.tscn):

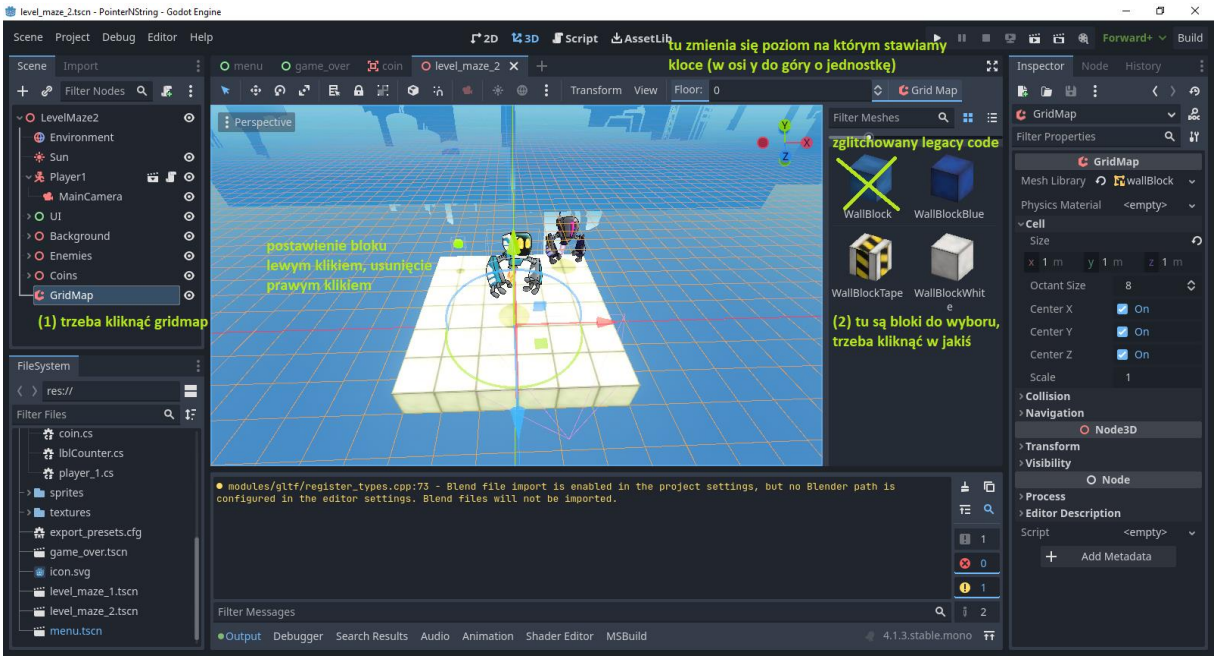




Jak chcecie odpalić tylko to to ten magiczny guzik do tego służy!

Szału nie ma ale na tejże scenie mamy absolutnie wszystkie elementy potrzebne do utworzenia nowego poziomu!

Edycja planszy:

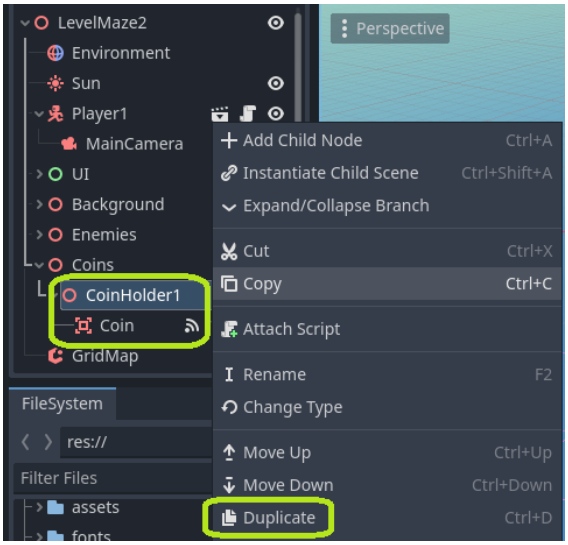


Można jak ja zrobić sobie drugi gridmap dla ścian labiryntu jak ja to wtedy prawym przyciskiem na GridMap w strukturze w zakładce Scene po lewej i Duplicate.

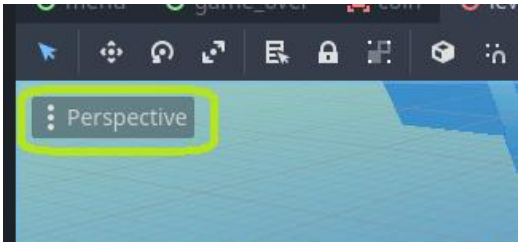
Dodawanie monet

To jest już trochę problematyczne bo skrypt do zliczania monet jest dzielony z sceną z poziomem1 więc 150 monet aktywuje wygraną. Można dopisać własną kopię skryptu i ją podpiąć lub to olać ;)))

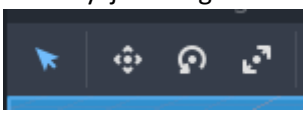
Generalnie monety są poumieszczane we własnych przestrzeniach 3D (ważne żeby działały ich animacje!) więc duplikując je trzeba skopiować całą paczkę



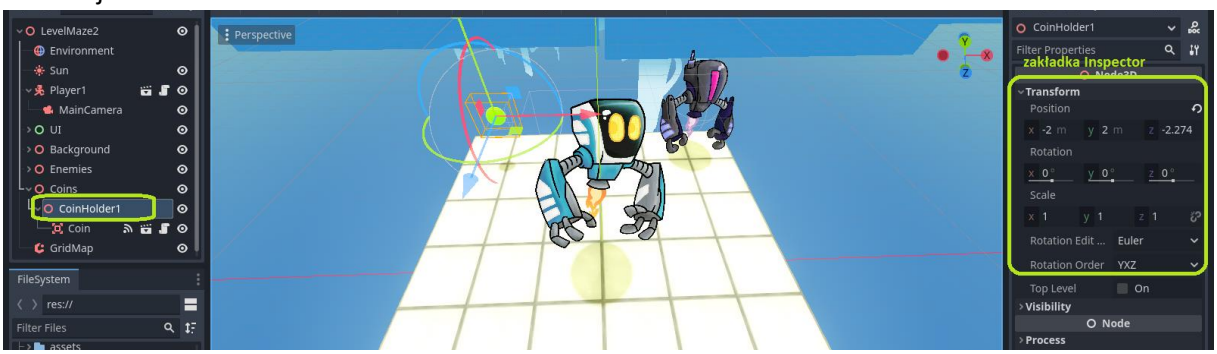
Przemieszczanie wszelkich obiektów najlepiej robić o ten najwyżej w danej hierarchii czyli tutaj CoinHolder**N.** Na numpadzie przyciski 7, 1 i 3 to widok od góry, przodu i boku planszy z osi x,y,z (w Godocie oś Y to pionowa do góry a nie z!!!) ale można też z menu rozwijanego:



I do edycji każdego elementu można użyć przemieszczenia, rotacji i skalowania co da się ustawić tutaj



lub tutaj

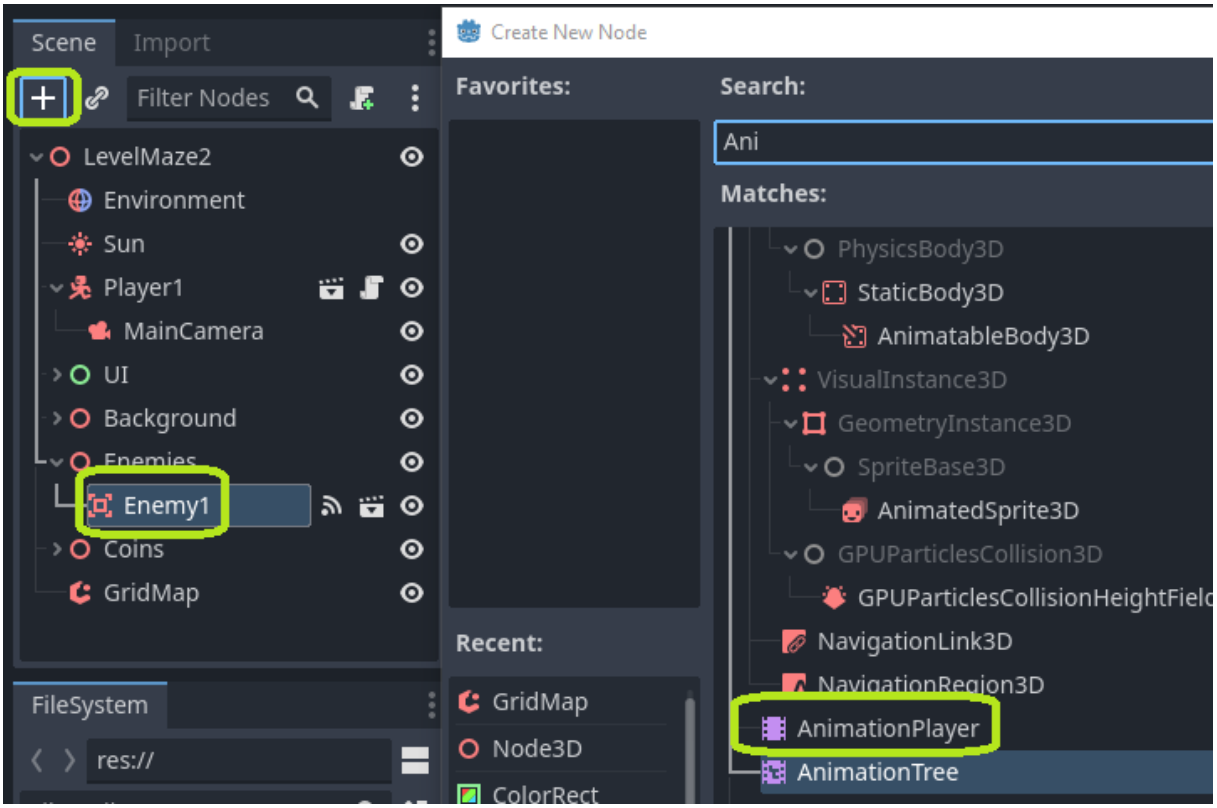


Wrogowie i programowanie ścieżek ruchu

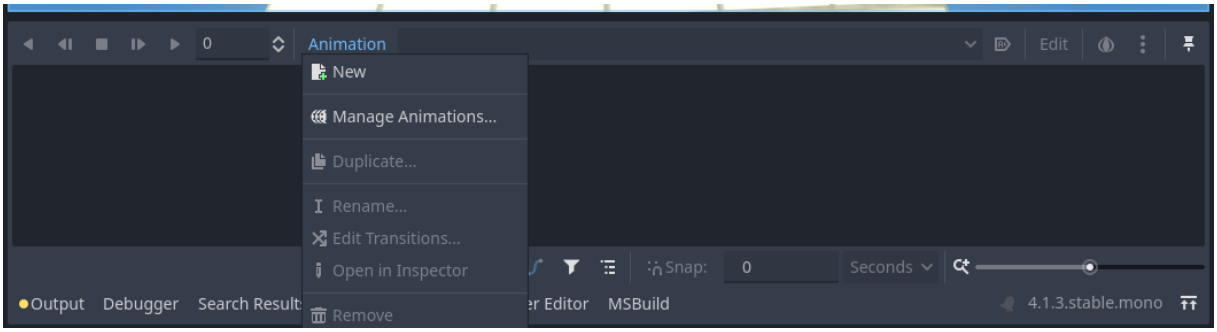
Wrogowie znajdują się w zakładce Enemies, tam też polecam ich kopiować

Można ich spokojnie przestawiać analogicznie do monet czy też gracza, kamery itd

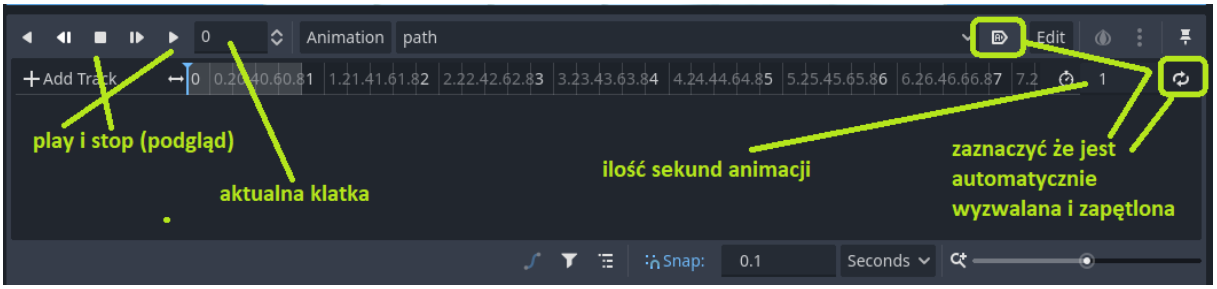
Aby zaprogramować ścieżke wrogowi trzeba kliknąć go w rozwijanym drzewie po lewej, kliknąć małą ikonę plusa i dodać instancję AnimationPlayer



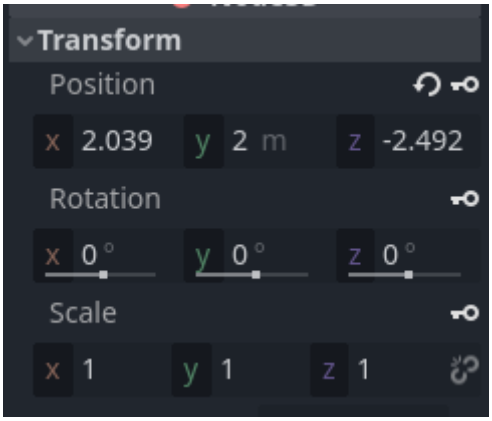
Klikamy w animationplayer (musi być w hierarchii po lewej dzieckiem tego wroga który ma się poruszać). Na dole powinien się otworzyć dok z animacją



New -> dajemy nazwę



Teraz żeby zapisać pozycję wroga na planszy trzeba w niego kliknąć w drzewie i pojawiają się takie małe kluczyki przy opcjach. Kliknięcie w jakikolwiek zapisuje w danej sekundzie tą wartość przekształcenia (przesunięcie, rotacja, skala) obiektu.



I to tyle, duplikacja klatek animacji to przejście na daną pozycję aktualnej klatki (np. 3.2) (w zasadzie to czas a nie klatki ale nie chce mi się już tego poprawiać), zaznaczyć co chcemy przekopiować (najlepiej od razu x,y,z) i ctrl+d.

Miłej zabawy, mam nadzieje że nie zanudziłem tym przydługawym pdf-em.