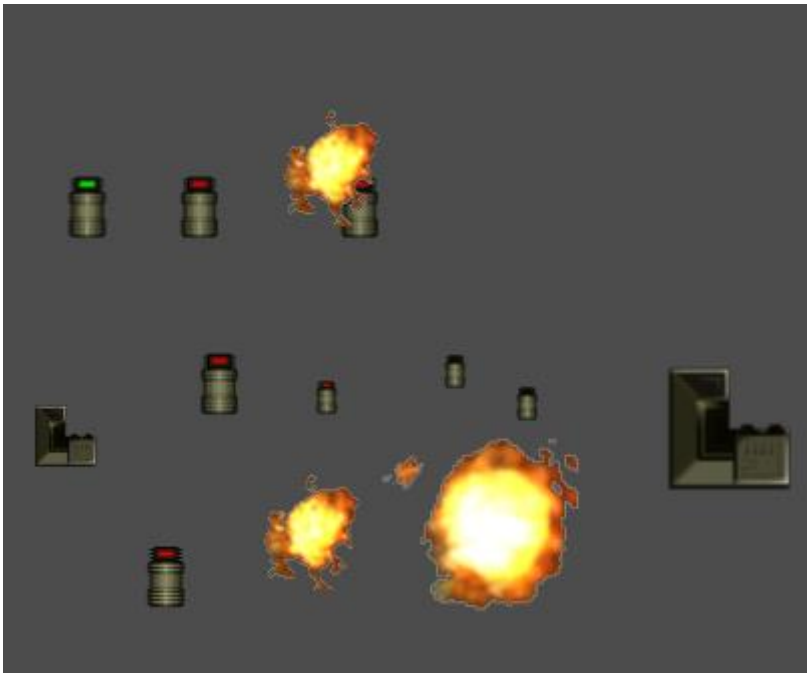
Szczecińskie Collegium Informatyczne
Szczecin – ul. Mazowiecka 13
www.sci.edu.pl

SZCZECIŃSKIE COLLEGIUM INFORMATYCZNE

www.acodemia.pl **e-mail:** info@acodemia.pl



Time bomb 2018 – 01 – 13

Zarządzanie czasem w grach komputerowych

Zarzadzanie czasem w grach komputerowych to jedno z najistotniejszych zadań projektantów i programistów gier komputerowych. Począwszy od uruchomiania gry, która jest niczym innym jak programem komputerowym – program działa w pętli czasu rzeczywistego.

Jest to pojęcie znane i stosowane przez programistów gier. Najogólniej rzecz ujmując, co jakiś czas (ułamek sekundy), gra komputerowa aktualizuje swój stan logiczny. Na przykład sprawdza, czy Gracz nie odniósł obrażeń, czy już czas, aby poinformować Gracza, że pora na posiłek, czy pocisk dotarł do celu, czy może trafił we wroga, itp. Jak łatwo zauważyć, procesor komputera ma ogrom pracy, zważywszy

na fakt, że przecież w grze, na ekranie widać czasami ogromną ilość obiektów, z których większość wymaga aktualizacji.

Po aktualizacji (aktualizacji logiki), gra „rysuje się na ekranie", a Gracz widzi, co się dzieje w danej chwili. Wszystko to przebiega w czasie rzeczywistym. Zauważ, że gra zużywa ogromną moc procesora na obliczenia stanów wszystkich obiektów gry. Zatem stan gry – logika musi „się wyrobić" w bardzo krótkim czasie. Na przykład w ciągu 1/60 sekundy, co daje liczbę `0,0166666666666667` sekundy. Jest to bardzo krótki czas. Jeśli weźmiemy teraz pod uwagę, że na scenie może być bardzo dużo obiektów i każdy z nich aktualizuje swój stan w ułamku sekundy – widać, że procesor może być „gorący" od obliczeń arytmetycznych. Gdy dodamy do tego ogrom pracy, jaką wykonuje karta graficzna – widać, że gra, która jest programem komputerowym wymaga ogromnego doświadczenia ze strony programistów i projektantów gier komputerowych. A przecież gra – program komputerowy musi jeszcze obsługiwać zdarzenia takie jak klawiatura, mysz, ekrany dotykowe, szereg innych oraz przetwarzać komunikaty generowane przez urządzenia, aby zapewnić Graczowi komunikację się z programem – grą.

Aby wszystko działało poprawnie, wyświetlanie grafiki i aktualizacja logiki, procesy te muszą być zsynchronizowane. Nie może przecież być takiej sytuacji, w której na ekranie widać, że pocisk dotarł do przeszkody, a program nie zdążył obliczyć, że nastąpiła kolizja! Albo karta graficzna narysowała eksplozję, a droga pocisku nie została jeszcze policzona, a sam pocisk „dopiero" zbliża się do przeszkody, czy celu.

Dla nas, najistotniejszą informacją oraz wnioskiem, jaki wynika z ogólnego pojęcia, jakim jest **pętla czasu rzeczywistego** jest fakt, że musimy zliczać upływ czasu w naszej grze. W środowisku **Godot Engine**, projektanci zadbali o to, aby proces zarządzania czasem był ułatwiony.

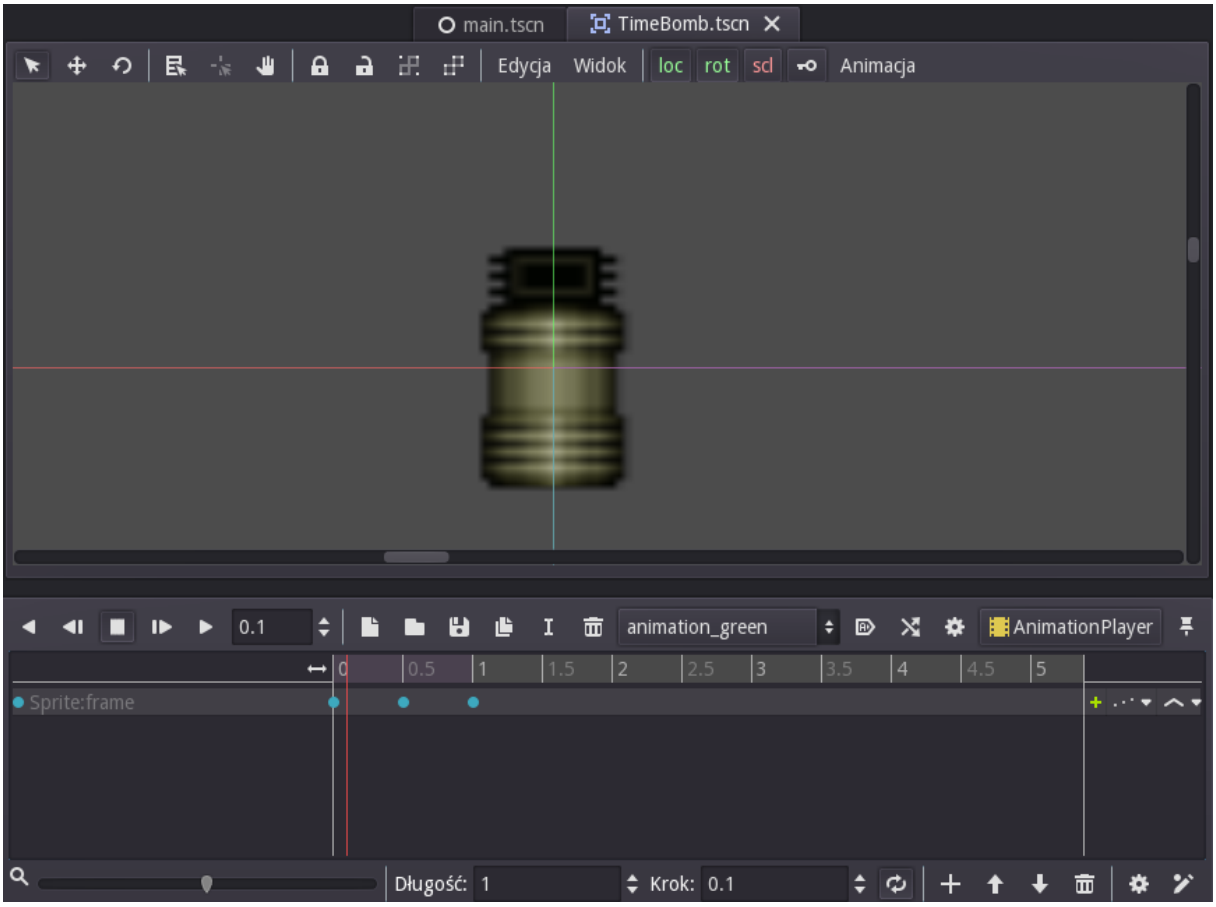
Szczecińskie Collegium Informatyczne
Szczecin – ul. Mazowiecka 13
www.sci.edu.pl

SZCZECIŃSKIE COLLEGIUM INFORMATYCZNE

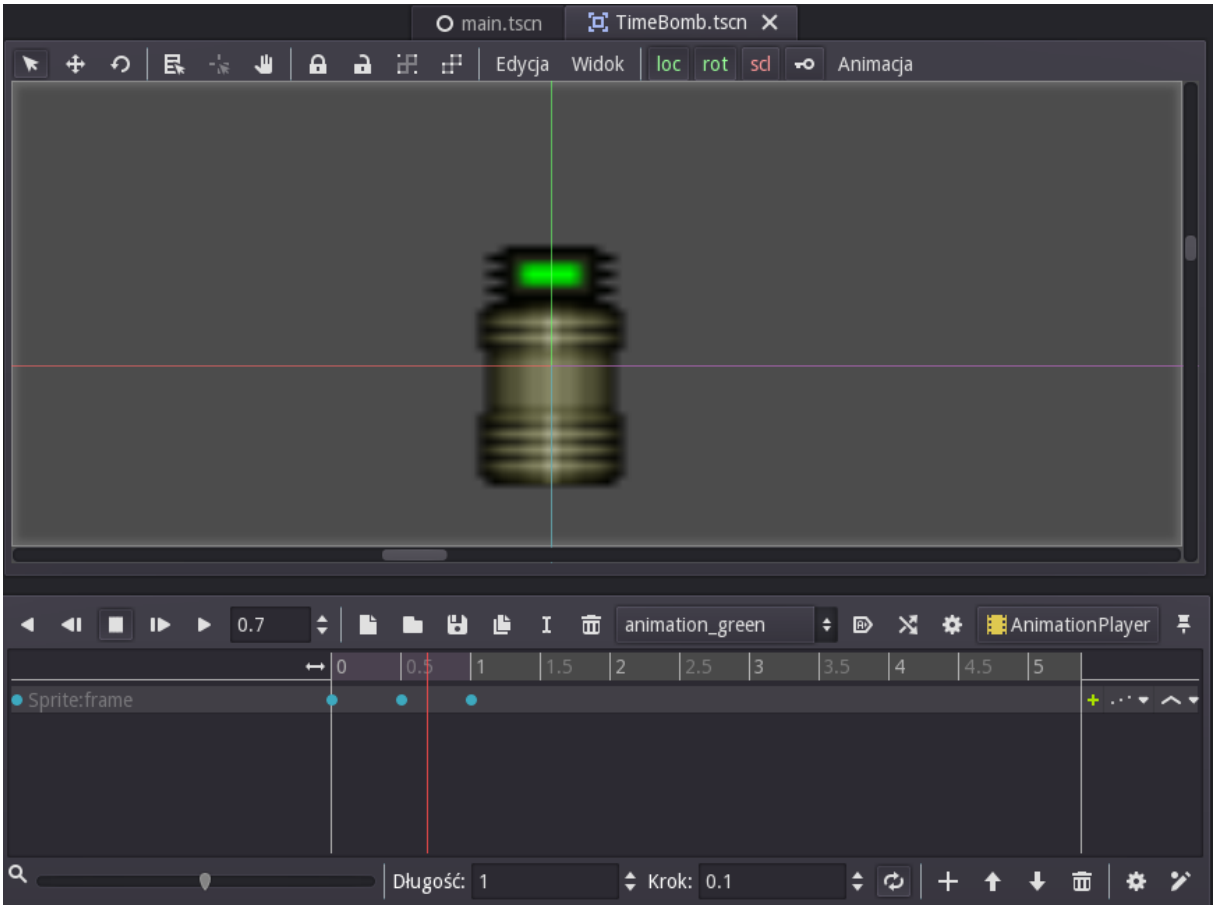
www.acodemia.pl **e-mail:** info@acodemia.pl

Pomysł

Naszym zadaniem jest uatrakcyjnienie gry poprzez zaprojektowanie bomby z opóźnionym zapłonem. Bomba, umieszczona na scenie przez pewien czas jest nieuzbrojona. Po upływie pewnego czasu, uzbraja się. Gdy ten czas minie, eksploduje, tym samym przestaje istnieć, ginie. Jeśli jakiś obiekt, np. Gracz, znajdzie w zakresie działania eksplozji – otrzymuje obrażenia, których wielkość zależy od rodzaju bomby. Warto dodać, że dla wizualizacji tego procesu, zastosowane zostały dwie animacje. Pierwsza to animacja migania zielonej kontrolki bomby (stan, gdy bomba nie jest jeszcze uzbrojona). Druga - animacja czerwonej kontrolki bomby (stan wskazuje, że bomba jest już uzbrojona i za jakiś czas eksploduje). O zastosowaniu i podstawowych opcjach tworzenia animacji (**AnimationPlayer**), powiemy w dalszej części tego opracowania. Proces „życia" bomby z opóźnionym zapłonem, przedstawiają poniższe zrzuty ekranowe.



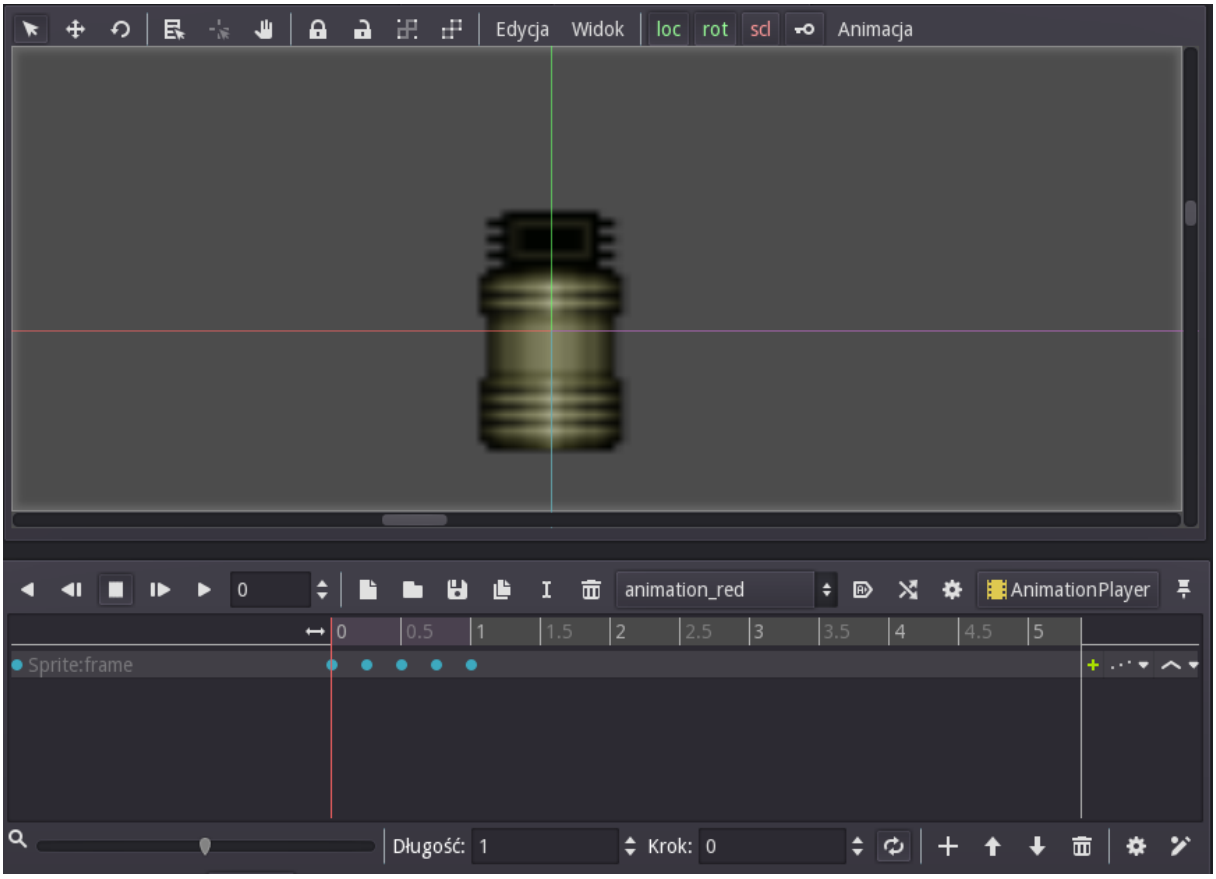
Rysunek 1 Bomba jest nieuzbrojona – zielona kontrolka jest zgaszona



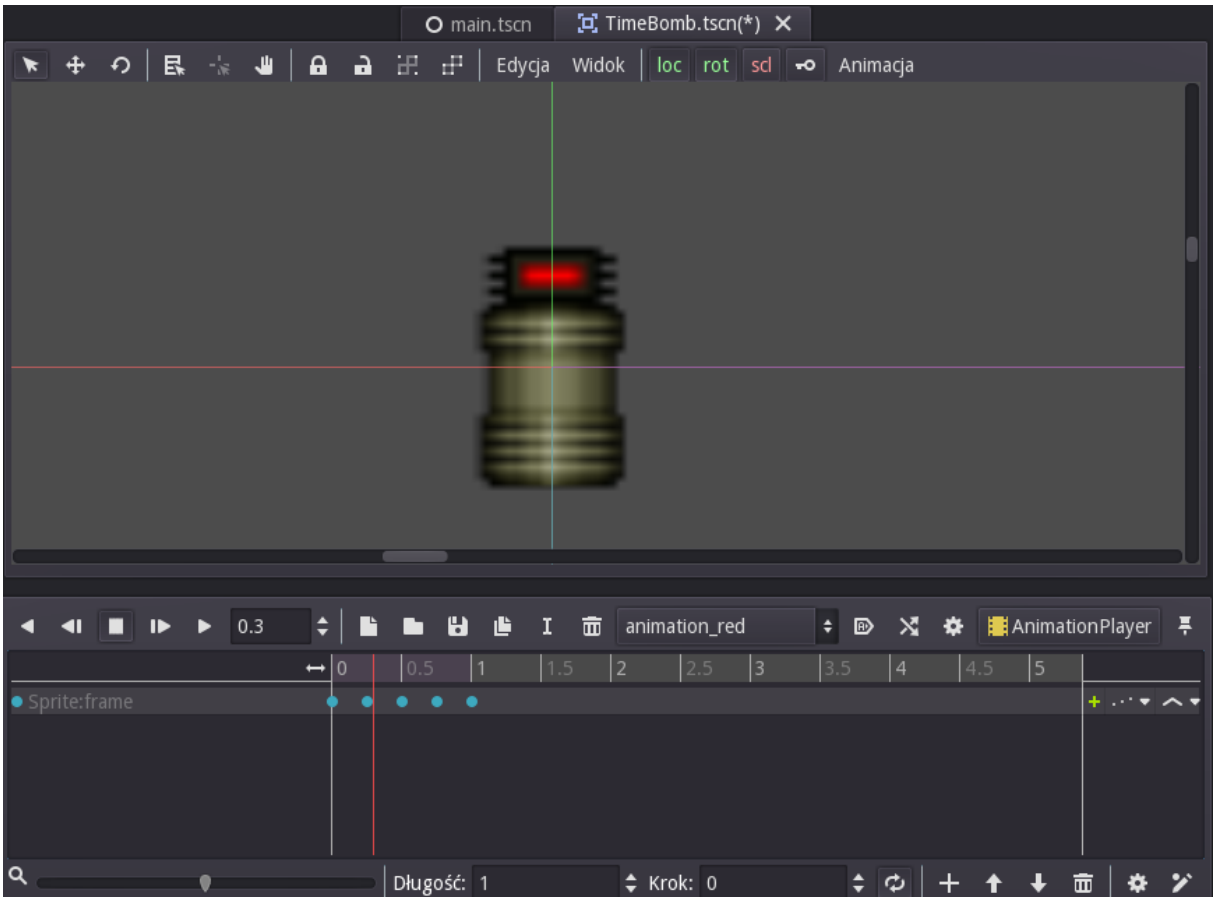
Rysunek 2 Bomba jest nieuzbrojona – zielona kontrolka jest aktywna

Odpowiednio dla stanu, gdy bomba jest uzbrojona – sygnalizuje to czerwony kolor kontrolki. Warto zauważyć, że obydwie animacje („zielona” i „czerwona”) mają taką samą długość – 1 sekunda, a różnica polega na ilości mignięć kontrolek. Kontrolka czerwona w ciągu jednej sekundy mignie 2 razy, czyli dwa razy więcej niż kontrolka zielona. Dzięki temu odniesiemy wrażeniem, że gdy bomba jest uzbrojona, czerwona kontrolka miga szybciej. Zastanawiając się, czy warto poświęcić tyle pracy, podczas gdy proces ten trwa tylko sekundy? Warto, ponieważ wszystkie te drobiazgi, smaczki, których jakość estetyczną potęguje ich ilość na scenie – podnoszą walory artystyczne

gry, a to jest jeden z istotniejszych aspektów, dla których Gracze wybierają takie, a nie inne produkcje.



Rysunek 3 Bomba jest uzbrojona – czerwona kontrolka jest wygaszona

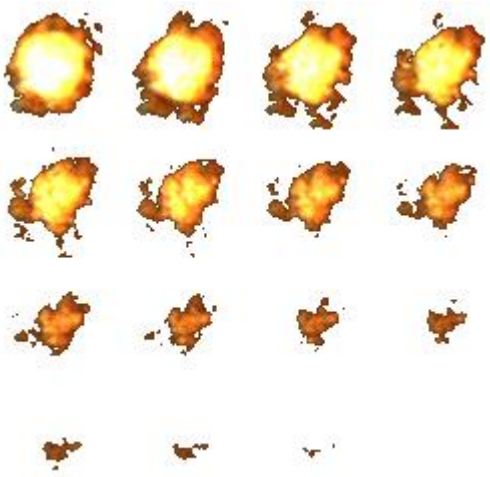


Rysunek 4 Bomba jest uzbrojona – czerwona kontrolka jest aktywna

Podsumujmy zatem, jakiej grafiki będziemy potrzebowali, aby zaprojektować taki obiekt – bombę?

Wystarczy, że przygotujemy jeden plik graficzny w formacje **PNG** z kanałem **Alpha**. Pliku, na którym znajduje się zbiór grafik (na przykład nasza bomba, czy kolejne klatki eksplozji), nosi nazwę atlasu tekstur. Grafik współpracując z programistą przygotowuje grafikę w odpowiedni sposób – umieszczając grafikę w atlasie tekstur Odpowiednio parametryzując te dane (teksturę) w **Godot Engine**, uzyskamy interesujące nas **Sprite**.

Rysunek 5 Atlas tekstur z grafiką bomby (do utworzenia animacji)



Rysunek 6 Atlas tekstur z klatkami animacji eksplozji bomby

Podsumowując można dodać, że w sieci Internet można odszukać gotowe atlasy tekstur. Wiele z nich jest darmowych. Taki atlas został wykorzystany w tym opracowaniu[1]. Znając rozmiar (w pikselach, szerokość i wysokość; 256 x 256 pikseli) tekstury oraz znając ilość klatek eksplozji, w naszym przypadku 16 łatwo wyliczyć, że klatka animacji będzie miała rozmiar 64 x 64 piksele.

Budujemy scenę

Wiemy, że mamy zaprojektować obiekt (scenę), która swoją funkcjonalnością będzie przypominała bombą czasową z opóźnionym zapłonem. Gracz, aby dostać się

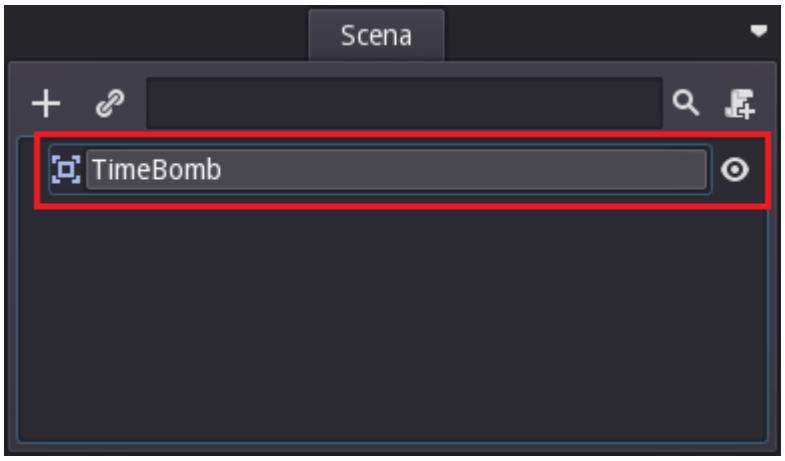
[1] Będzie o tym traktował osobny artykuł o tworzeniu sceny eksplozji.

do sceny, na której znajduje się jego pojazd[2], będzie musiał pokonać pole, na którym umieścimy szereg bomb z opóźnionym zapłonem.

Docelowo, tak zaprogramujemy bomby, aby wybuchały w sposób losowy (pseudolosowy), co uatrakcyjni naszą rozgrywkę[3]. Powtórne uruchomienie gry i pokonanie sceny z bombami, będzie za każdym razem inne - niepowtarzalne.

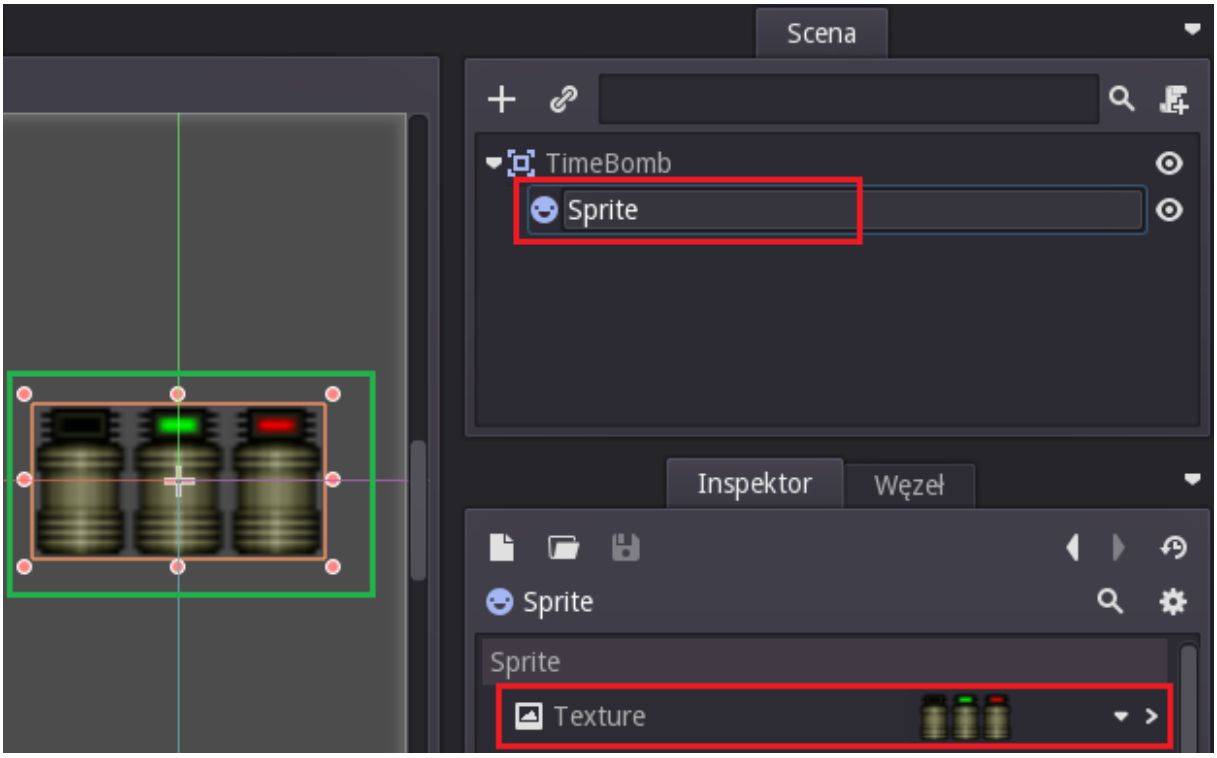
Projektanci i programiści komputerowi od zawsze starają się tak budować swoje gry, aby każda rozgrywka była unikalna, niepowtarzalna. Projektowane światy sprawiają wrażenie światów otwartych, nieskończonych. Zdarzenia umieszczane w obrębie tych światach wydają się dla Gracza unikalne, niepowtarzalne. Wszystkie te zabiegi stosuje się po to, aby zapewnić Graczowi jak najlepszą rozrywkę, zabawę. Nic tak nie bawi jak zaskoczenie Gracza. Dlatego każdorazowe pokonanie labiryntu, ukończenie zadania, misji, pokonanie potwora, powinno być inne.

1. Z menu projektu **Scena** wybieramy **Nowa Scena**. W oknie sceny dodajemy węzeł główny, który jest typu **Area2D**, a następnie zmieniamy jego nazwę na **TimeBomb**.

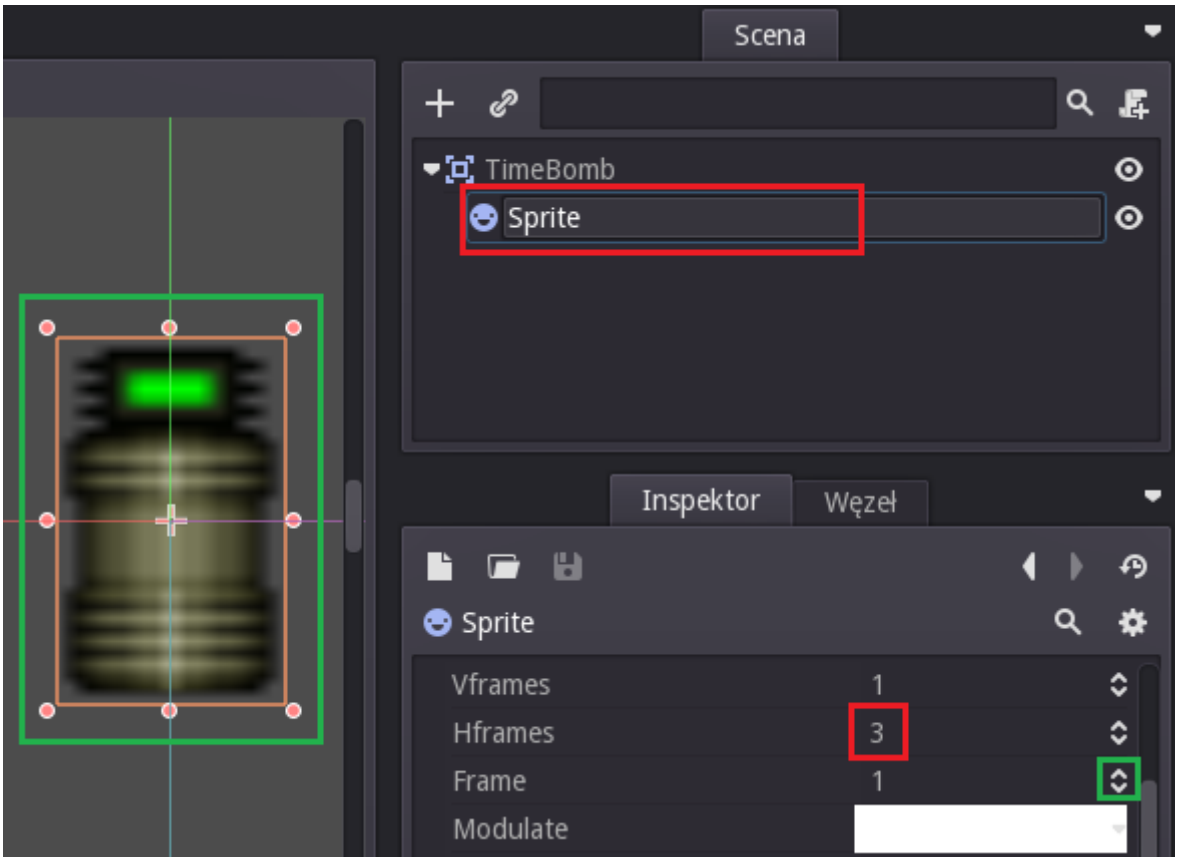


[2] W projekcie wykorzystującym środowisko Godot Engine, na zajęciach w ramach projektu **Acodemia++**, jest to scena, na której umieściliśmy pojazd kosmiczny **Starship**.
[3] O mechanizmie pseudolosowości w programowaniu będzie osobny artykuł – opracowanie.

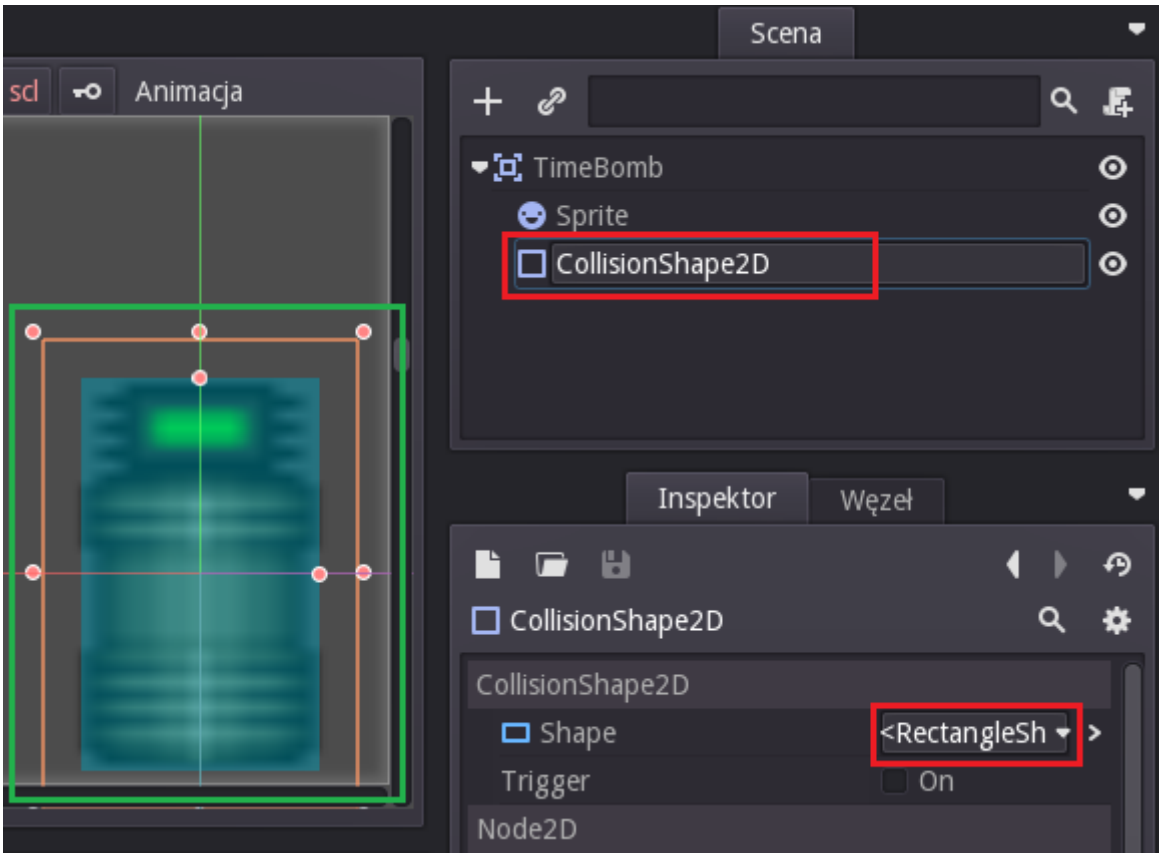
2. Zaznaczamy węzeł **TimeBomb** i znakiem **+** dodajemy do tego węzła węzeł **Sprite**. Następnie za pomocą **Inspektora** ładujemy z dołączonego do projektu folderu **timebomb** teksturę z grafiką bomby, co prezentuje poniższy zrzut ekranowy.



3. Teraz tak skonfigurujemy teksturę wgraną na węzeł **Sprite**, aby „pociąć" ją na trzy elementy. Zaznaczamy węzeł **Sprite**, a w **Inspektorze** znajdujemy opcję **Hframes** (ilość klatek w poziomie) i zmieniamy wartość z 1 na 3. Silnik **Godot Engine** już samodzielnie podzieli teksturę, przeliczy ilość klatek oraz zmieni wygląd obiektu na scenie. Aby przetestować, czy nasze działania są poprawne, można to sprawdzić, zmieniając w Inspektorze opcję **Frame** w zakresie od, 0 do 2 – czyli trzy klatki. Efekt powinien być widoczny na scenie. Poniżej efekt zmian.

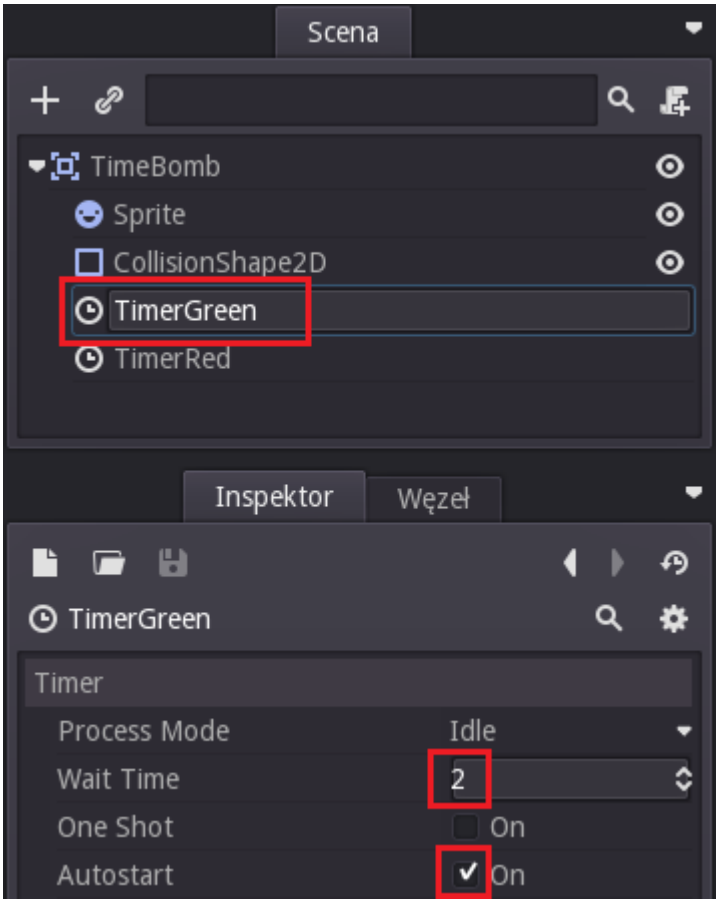


4. Zaznaczamy węzeł główny sceny **TimeBomb** i dodajemy węzeł reprezentujący kolizję. W odniesieniu do tego węzła wybieramy rodzaj kolizji, a następnie dopasowujemy jej rozmiar do rozmiaru grafiki **Sprite**. Prezentuje to poniższy zrzut ekranowy. Kolizja będzie przydatna, gdy będziemy chcieli w przyszłości wprowadzić funkcjonalność reakcji bomby na samą kolizję. Możemy na przykład zaprogramować sytuację, gdy saper chce taką bombę rozbroić, podchodzi do niej, a ta eksploduje (lub nie, o czym mogłaby zdecydować sztuczna inteligencja – prosty skrypt).



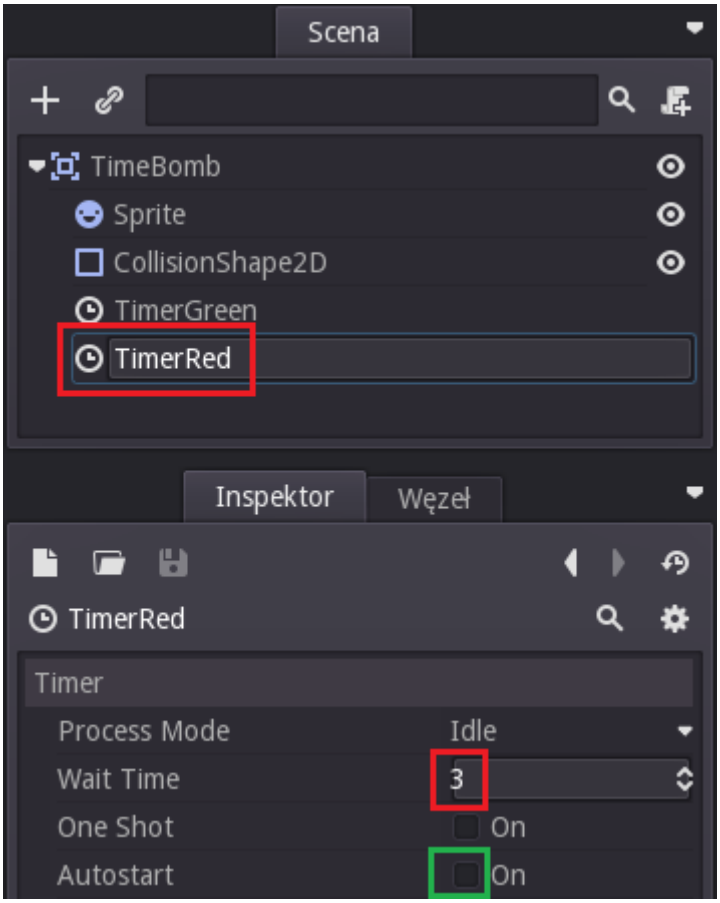
5. Dodamy teraz do naszej sceny dwa zegary (**Timer**). Pierwszemu przypiszemy nazwę **TimerGreen**, drugiemu **TimerRed**. Jak można się łatwo domyśleć, pierwszy będzie zliczał czas do uzbrojenie bomby, drugi do jej eksplozji. Na potrzeby tego opracowania zegar „zielony”[4] ma ustawiony czas na 2 sekundy, zegar czerwony jest ustawiony na 3 sekundy. Ważne jest, aby zegar zliczający czas „zielony” miał ustawioną flagę autostartu. Spowoduje to, że zacznie odliczać czas zaraz po uruchomieniu sceny. Natomiast zegar „czerwony” nie ma tego ustawienia. Jest to istotne, ponieważ najpierw musi minąć czas, w którym bomba nie jest uzbrojona. Dopiero po jego upływie, zatrzymamy ten

zegar, a uruchomimy zegar „czerwony”. Wszystko to będzie miało swoje odzwierciedlenie w kodzie skryptu, który dodamy do tej sceny. Poniżej dwa zrzuty ekranowe, na których zaznaczono ustawienia poszczególnych zegarów.



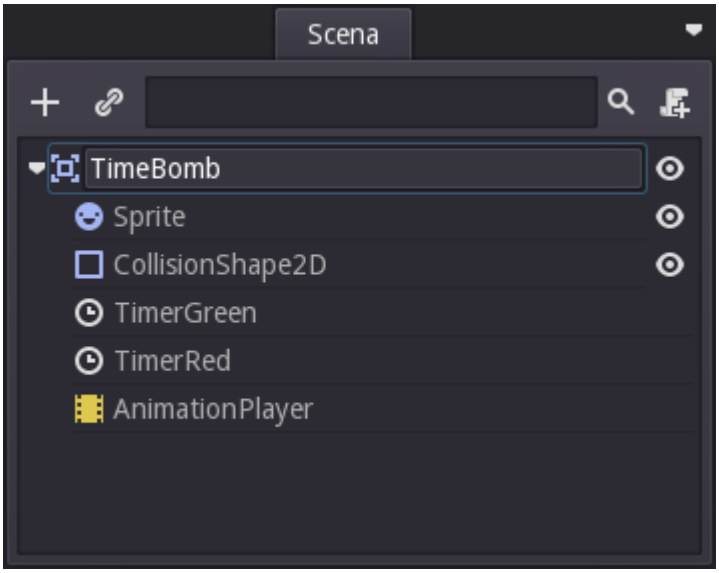
Rysunek 7 Ustawienie zegara "zielonego"

[4] Umawiamy się, że to określenie odnosi się do stanu bomby, w którym jest ona nieuzbrojona. Przypisaliśmy temu kolor (znaczenie) zielony. Odpowiednio kolor czerwony to stan, w którym bomba jest już uzbrojona i za jakiś czas wybuchnie.



Rysunek 8 Ustawienie zegara "czerwonego"

6. Dodamy teraz do naszej sceny węzeł reprezentujący animację. Zaznaczamy węzeł główny **TimeBomb**, wciskamy ikonę „**+**” i znajdujemy obiekt (węzeł) **AnimationPlayer**. Kompletna scena powinna zawierać węzły w konfiguracji, którą przedstawia poniższy zrzut ekranowy.



7. Zapisujemy scenę pod nazwą **TimeBomb.tscn** do folderu **timebomb**.

Animacja

Zajmiemy się teraz pracą z edytorem animacji, za pomocą którego zaprojektujemy dwie animacje bomby. Pierwsze będzie animacją migania bomby – naprzemienna zmiana klatki na kolor zielony, druga – na kolor czerwony. Dodatkowo tak zaprojektujemy te animacje, aby „miganie zielone”[5] sprawiało wrażenia wolniejszego od „migania czerwonego”.

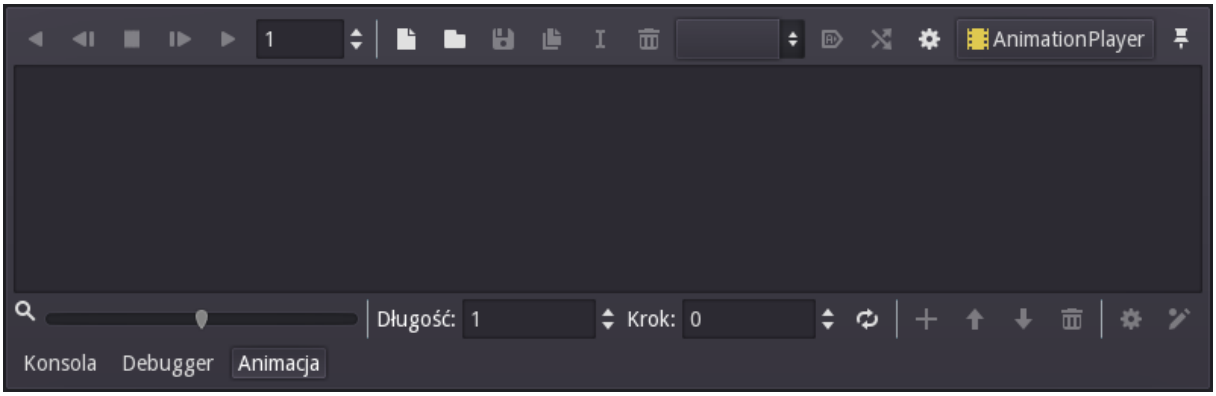
Dla sceny **TimeBomb**, zaznaczamy węzeł **AnimationPlayer**. Pojawi się panel, za pomocą którego można zarządzać animacjami. Na szczególną uwagę zasługuje fakt, że w środowisku **Godot Engine**, animować możemy praktycznie większością obiektów (scen i ich składowych). Na przykład można animować transformacją, czyli położeniem, obrotem oraz skalą. Można animować kolorem obiektów posiadających reprezentację graficzną, na przykład **Sprite** – przejścia

kolorów, przezroczystość, itp. Można tworzyć bardzo złożone animacje, a odpowiednio je eksportując, wykorzystywać w wielu projektach. W ramach tego opracowania utworzymy animację – efekt migania bomby.

Nasze opracowanie nie wyczerpuje wszystkich możliwości wykorzystania modułu tworzenia i zarządzania animacją. Skupimy się jedynie na zbudowaniu animacji migania bomby, wykorzystując poniższą sekwencję wymaganych czynności. Warto eksperymentować, aby poznać zasady zabawy z animacją w środowisku **Godot Engine**.

1. Zaznacza węzeł AnimationPlayer. 

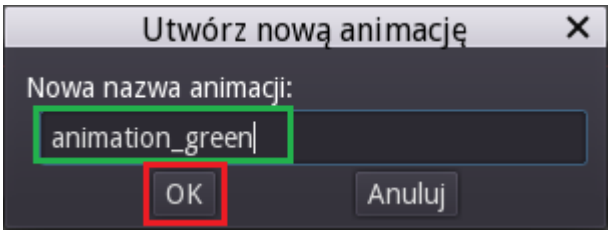
2. Zostanie aktywowany panel do tworzenia i zarządzania animacją.



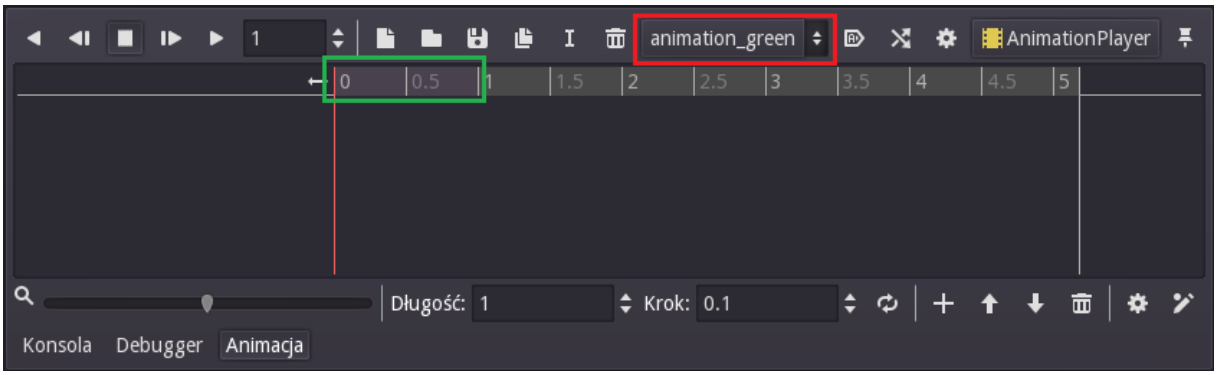
3. Najpierw utworzymy nową animację. Wciskamy ikonę reprezentującą tworzenie nowej animacji  .

[5] Pojęcia zastosowane dla uproszczenia treści tego opracowania

Nadajemy jej nazwę **animation_green** i zatwierdzamy.

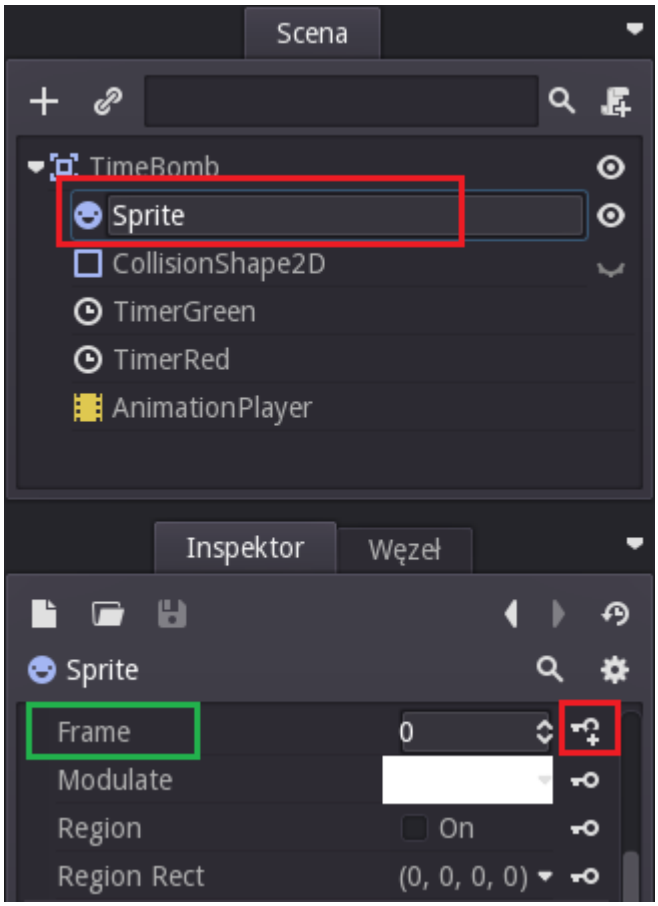


Animacja została zarejestrowana, co jest widoczne w panelu do animacji. Na zrzucie ekranowym kolorem czerwonym zaznaczono, że będziemy tworzyli animację, której nadaliśmy nazwę **animation_green**, a zaznaczenie w kolorze zielonym informuje, że czas trwania animacji (pełnego obiegu), to 1 sekunda. Zostawiamy ustawienia domyślne.



4. Będziemy animowali zmianą klatek, czyli grafiką **Sprite** bomby. Zaznaczamy w oknie sceny węzeł **Sprite**. Następnie w Inspektorze odszukujemy **Frame**. Obok wpisów cech, które modyfikujemy w Inspektorze, pojawiły się ikony z kluczem . Klucz obok **Frame** jest odrobinę inny , ale jego działanie

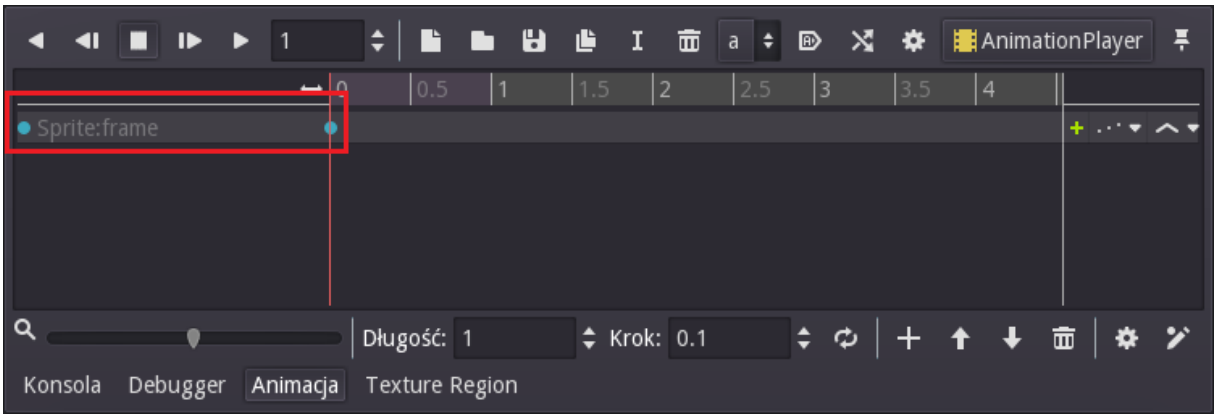
podobne. Warto wiedzieć, że jeśli do sceny dodamy węzeł **AnimationPlayer**, w Inspektorze, obol pojawią się ikony kluczy.



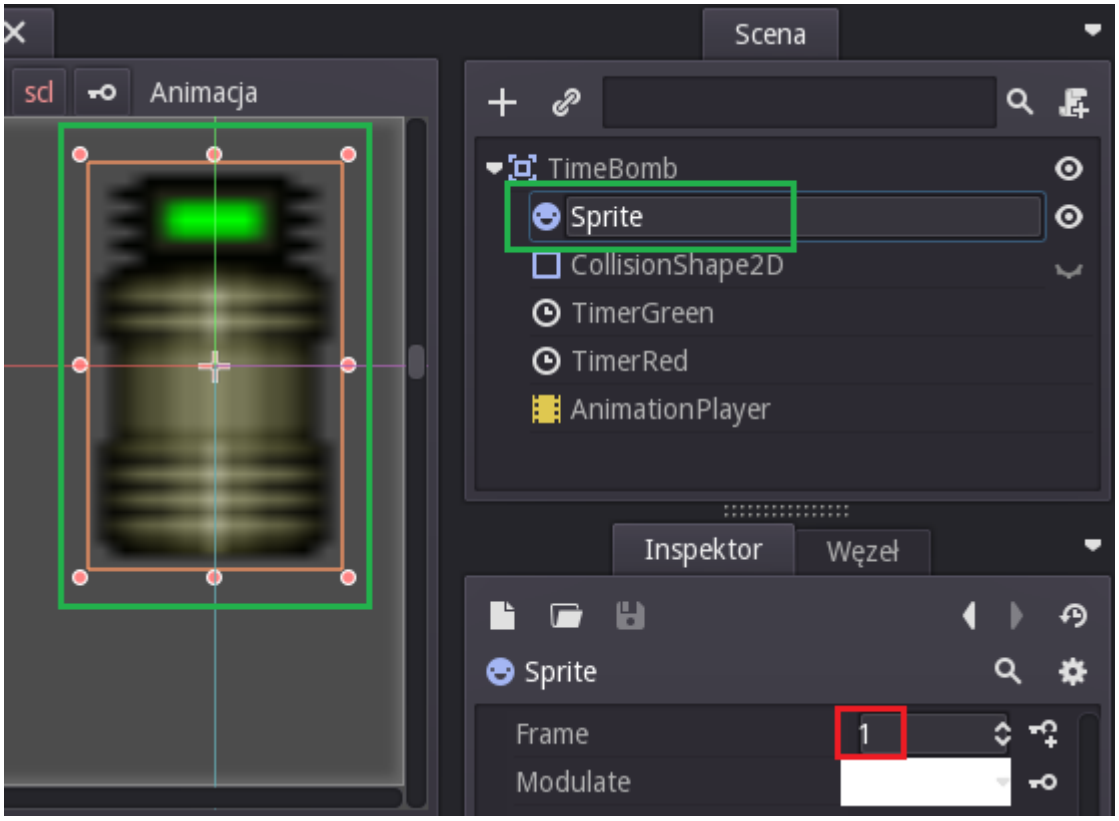
5. Upewniamy się, że wartość wpisu przy **Frame** wynosi **0** i wciskamy klucz  .

6. Pojawi się okno informujące, że należy zarejestrować ścieżkę animacji, odpowiedzialną za zarządzanie zmianami klatek (grafiką) **Sprite**.

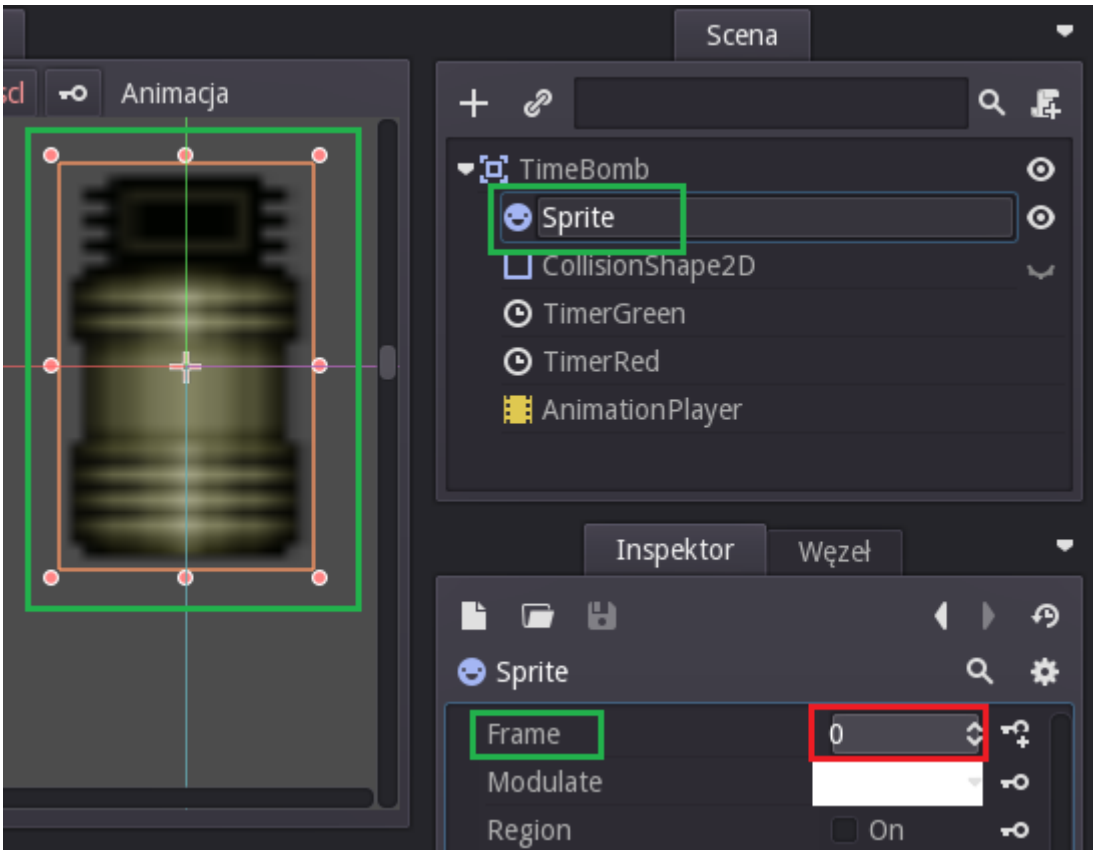
7. Ścieżka została zarejestrowana na liście.



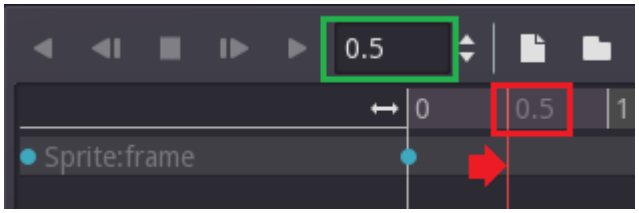
8. Zauważ, że automatycznie wartość Frame zmieniła się na 1, a na scenie nasza bomba „zapaliła” się na „zielono.



9. **Cofamy** wartość Frame z powrotem, na 0 ponieważ chcemy, aby animacja rozpoczęła się od „wygaszonej kontrolki” i wciskamy klucz.

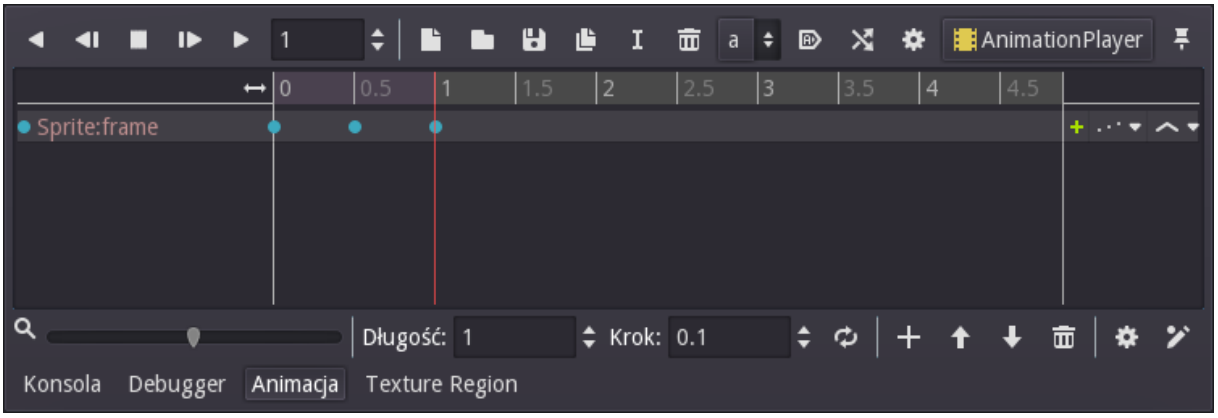


10. Przesuwamy na osi czasu animacji wskaźnik, aby pokazywał połowę czasu animacji, czyli 0.5 sekundy.



11. Zmieniamy wartość **Frame** na 1 – obraz Spite na scenie zmieni się na klatkę z kolorem zielonym. Wciskamy klucz przy **Frame**.

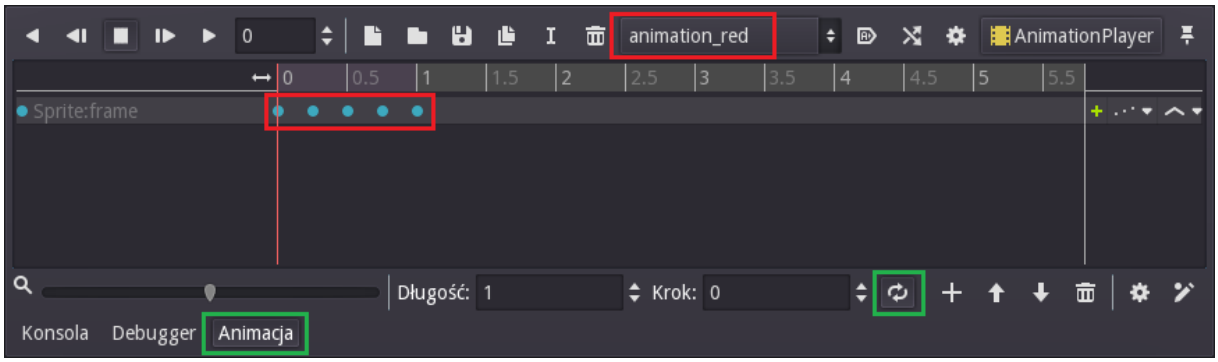
12. Przesuwamy się na koniec osi czasu – 1 sekunda, cofamy Frame na 0 i ponownie naciskamy klucz. Na panelu powinniśmy posiadać następującą konfigurację.



13. Uruchamiamy animację od początku wciskając ikonę oraz sprawdzamy jak można odtwarzać animację testując pasek z narzędziami do jej odtwarzania .

14. Włącz zapętlenie animacji za pomocą ikony znajdującej się w dolnej części panelu animacji. Jeśli w oknie sceny, na której jest bomba, co pół sekundy bomba „miga” na zielono – oznacza to, że wszystko zostało zaprojektowane poprawnie.

Bazując na powyższych informacjach – tworzenie animacji „migania” bomby w kolorze zielonym, samodzielnie utwórz animację, która powoduje, że bomba będzie „migała” w kolorze czerwonym. Spróbuj zaprojektować ją w taki sposób, aby w czasie 1 sekundy bomba „migała” dwa razy. Poniżej zrzut, który może posłużyć, jako podpowiedź. Zauważ, że zarejestrowaliśmy dla węzła **AnimationPlayer** animację o nazwie **animation_red**.



Opanowanie umiejętności tworzenia animacja wymaga czasu. Warto dodać, że w różnych środowiskach do projektowania i programowania, zastosowane narzędzia do animacji działają na podobnej zasadzie. Pamiętaj, że tworzenie poszczególnych ujęć animacji opiera się na następującej zasadzie:

- Wybierz obiekt, którym będziesz animował
- Wybierz, co w danym obiekcie będzie podlegało animacji
- Zaznacz na osi czas, zmodyfikuj obiekt – zarejestruj (klucz)

Pamiętaj, aby zapisać scenę **TimeBomb**.

Dodajemy bombę do sceny głównej

Przejdź do sceny głównej, a następnie dodaj do niej węzeł reprezentujący bombą z opóźnionym zapłonem – **TimeBomb**, posługując się mechanizmem **instancjonowania**.

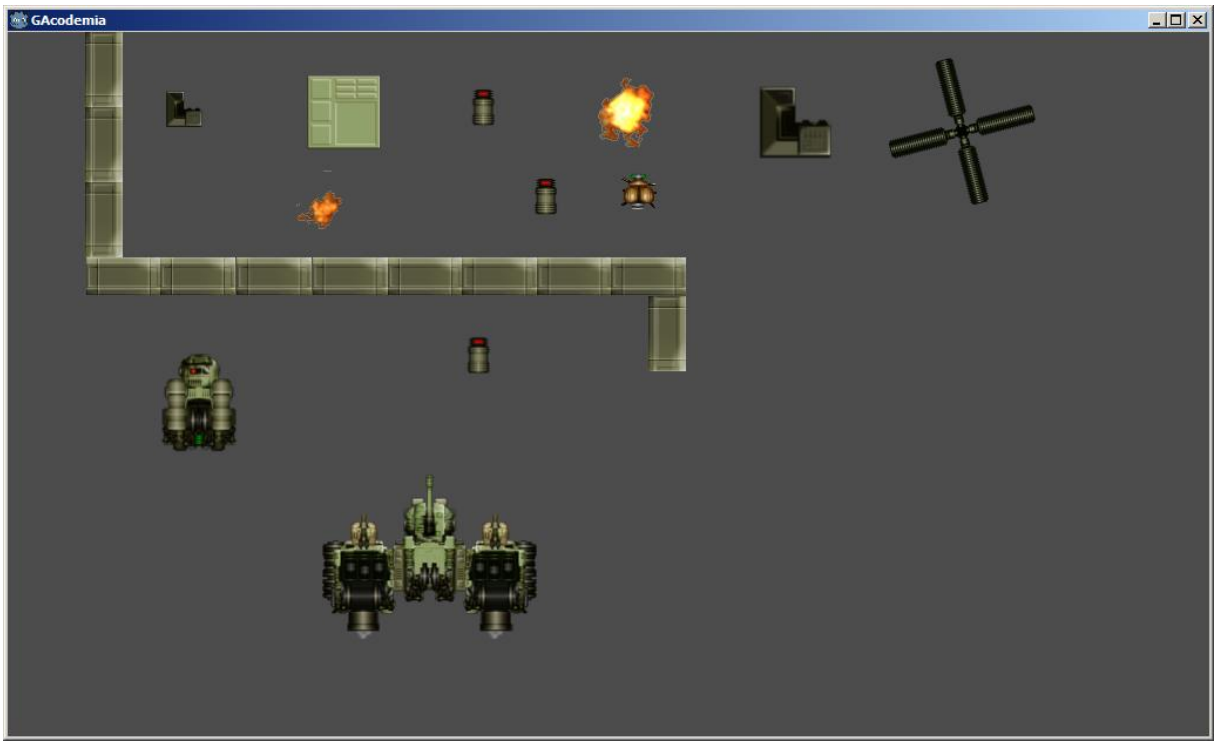
Niniejsze opracowanie nie wyczerpuje wszystkich zagadnień realizowanych na zajęciach projektu Acodemia++. W obrębie tych zajęć Uczestnicy otrzymują częściowo przygotowane obiekty, grafikę, gotowe skrypty, dane, wszelkie inne.

Szczecińskie Collegium Informatyczne
Szczecin – ul. Mazowiecka 13
www.sci.edu.pl

SZCZECIŃSKIE COLLEGIUM INFORMATYCZNE

www.acodemia.pl **e-mail:** info@acodemia.pl

Poniżej zrzut ekranowy z działającym programem.



Zespół Acodemia++