Horrorek: poprawki i pułapki

Naprawione błędy w formacie: objaw, przyczyna, rozwiązanie, test, zadanie.

- Drzwi: kolizja skrzydła zostawała w pozycji zamkniętej
- Dialogic: brak akcji wejścia dialogic_default_action
- Questify: autoload i ustawienia przy pierwszym uruchomieniu
- Phantom Camera: sprawdzanie aktualizacji przy starcie edytora
- PowerShell 5.1: tekst na stderr przerywa skrypt testów
- Esc na ekranie awansu lub zapisu zamykał też menu pod spodem
- Śmierć w trakcie przejścia: gracz wstawał w miejscu śmierci

Drzwi: kolizja skrzydła zostawała w pozycji zamkniętej

Drzwi: kolizja skrzydła zostawała w pozycji zamkniętej

Objaw

Drzwi po otwarciu kluczem wizualnie się obracały, ale gracz nie mógł przejść przez otwór. Pomiar promieniami pokazał, że w całym otworze promienie trafiają w skrzydło (`Panel`), a prześwit przejścia wynosi 0 m. Kapsuła gracza zatrzymywała się w tym samym miejscu co przy zamkniętych drzwiach.

Przyczyna

W scenie `core/interaction/door.tscn` tween obracał zwykły węzeł `Pivot` (Node3D), a skrzydło `Panel` (AnimatableBody3D) było jego dzieckiem. AnimatableBody3D przenosi do silnika fizyki tylko ruch, który wykonuje sam. Zmiana transformacji rodzica nie aktualizowała położenia ciała w fizyce, więc mesh się obracał, a kolizja stała w pozycji zamkniętej.

Wcześniejszy test (`test_interaction.gd`) sprawdzał tylko kąt obrotu węzła, a nie to, czy da się przejść, więc błąd przeszedł niezauważony.

Rozwiązanie

- Zawiasem jest teraz sam `AnimatableBody3D` (węzeł `Hinge`) z `sync_to_physics = true`. To jego obraca tween (`rotation:y`).
- Kształt kolizji (BoxShape3D 1.2 × 2.2 × 0.1) i mesh są dziećmi zawiasu, przesuniętymi o pół szerokości skrzydła.
- Tween działa w kroku fizyki (`Tween.TWEEN_PROCESS_PHYSICS`).
- API drzwi się nie zmieniło: `required_item`, `open_flag`, sygnały `denied` i `opened`, `is_open`, start w pozycji otwartej przy ustawionej fladze.

Zasada na przyszłość: ruchome ciało fizyczne obracamy bezpośrednio, a nie przez węzeł nad nim.

Test

`tests/test_door_passage.gd` działa na prawdziwej fizyce i geometrii pokoju z drzwiami (dziś `content/rooms/room_b.tscn`; pierwotnie pokój testowy `room_test.tscn`, usunięty w zadaniu 005B):

- kapsuła 0.35 / 1.7 m z `move_and_slide` idzie przez otwór z trzech torów (środek i ±0.35 m),
- przy zamkniętych drzwiach zatrzymuje się przed otworem, a po `Door.open()` przechodzi na drugą stronę,
- promienie mierzą otwór (1.20 × 2.20 m) i prześwit przy otwartym skrzydle (1.13 m przy kącie 100°, wymagane co najmniej 1.0 m).

Przed poprawką test pokazywał prześwit 0.00 m i zablokowane przejście na wszystkich torach.

Zadanie

003A (poprawki i sterowanie), gałąź `agent/claude/faza1-poprawki`.

Opublikowano z repozytorium Horrorek (commit 87717ad) skryptem publikuj-bookstack.ps1.

Dialogic: brak akcji wejścia dialogic_default_action

Dialogic: brak akcji wejścia dialogic_default_action

Objaw

Podczas gry Dialogic zgłaszał błąd z `subsystem_input.gd`: `The InputMap action "dialogic_default_action" doesn't exist`. Dialogi nie reagowały na przycisk przewijania.

Przyczyna

Dialogic tworzy akcję `input/dialogic_default_action` tylko w `_enable_plugin()` (funkcja `add_dialogic_default_action()` w `addons/dialogic/plugin.gd`). Ta funkcja uruchamia się, gdy włączamy wtyczkę w oknie ustawień projektu. W Horrorku wtyczka jest włączona wpisem w `project.godot` (sekcja `[editor_plugins]`), bez klikania w edytorze, więc akcja nigdy nie powstała.

Rozwiązanie

Akcja jest wpisana do sekcji `[input]` w `project.godot` z tymi samymi zdarzeniami co w `add_dialogic_default_action()`: Enter, lewy przycisk myszy, Spacja, X, przycisk A pada, martwa strefa 0.5. Blok wygenerował sam Godot. Skrypt ustawił akcję przez `ProjectSettings.set_setting(...)`, zapisał ją przez `ProjectSettings.save_custom(...)` do pliku tymczasowego, a stamtąd blok trafił do `project.godot`. Wtyczka (`addons/`) została bez zmian.

Uwaga: w zapisie klawisze mają `device: 16`, a przycisk pada `device: 0`, tak samo jak w kodzie wtyczki. [DO UZUPEŁNIENIA: sprawdzić na prawdziwym padzie, czy działa także drugi kontroler.]

Test

`tests/test_plugins.gd` sprawdza `InputMap.has_action("dialogic_default_action")` i kończy się kodem 1, gdy akcji brakuje. Kontrola negatywna: na `project.godot` sprzed poprawki test wypisuje `ERROR input dialogic_default_action: MISSING`.

Zadanie

001b, PR #5.

Opublikowano z repozytorium Horrorek (commit 87717ad) skryptem publikuj-bookstack.ps1.

Questify: autoload i ustawienia przy pierwszym uruchomieniu

Questify: autoload i ustawienia przy pierwszym uruchomieniu

Objaw

1. Na świeżym projekcie import kończył się błędami `Identifier "Questify" not declared`.
2. Po obejściu pierwszego błędu gra przy starcie kończyła się błędem `Invalid cast: could not convert value to 'bool'`.

Przyczyna

1. Pliki wtyczki odwołują się do autoloadu `Questify`, który wtyczka rejestruje dopiero sama, przy włączeniu w edytorze. Projekt włącza wtyczki wpisem w `project.godot`, więc przy pierwszym imporcie autoloadu jeszcze nie było.
2. Wtyczka czyta ustawienia `questify/general/*` jako wartości logiczne i liczby. Bez tych kluczy w `project.godot` odczyt zwracał wartość, której nie dało się zamienić na `bool`.

Rozwiązanie

W `project.godot`:

- wpis autoloadu `Questify` w sekcji `[autoload]` (skrypt `addons/questify/quest_manager.gd`), dodany przed pierwszym uruchomieniem,
- sekcja `[questify]` z wartościami domyślnymi: `general/update_polling=true`, `general/update_interval=0.5`, `general/add_quests_to_pot_generation=false`.

Test

`tests/test_plugins.gd` wypisuje `autoload Questify: OK` i listę sygnałów Questify. Import (`--import`) i start edytora (`--editor --quit`) kończą się bez błędów (zob. `docs/krok-zero-raport.md`).

[DO UZUPEŁNIENIA: plik i linia wtyczki, w których pada rzutowanie na bool.]

Zadanie

Krok zero, PR #2.

Opublikowano z repozytorium Horrorek (commit 87717ad) skryptem publikuj-bookstack.ps1.

Phantom Camera: sprawdzanie aktualizacji przy starcie edytora

Phantom Camera: sprawdzanie aktualizacji przy starcie edytora

Objaw

Przy starcie edytora Phantom Camera zgłaszała błąd typów: zamiast tablicy (Array) dostawała słownik (Dictionary). Błąd pojawiał się bez dostępu do internetu albo po przekroczeniu limitu zapytań.

Przyczyna

Wtyczka przy starcie edytora pyta serwis GitHub o nowe wersje. Gdy zamiast listy wydań przychodzi odpowiedź z błędem (słownik z komunikatem), kod, który oczekuje tablicy, się wywraca. W środowisku z ograniczonym ruchem wychodzącym (CI, agent) wtyczka potrzebowałaby dostępu do `api.github.com`.

Rozwiązanie

W `project.godot` w sekcji `[phantom_camera]` ustawiamy `updater/updater_mode=0`, co wyłącza sprawdzanie aktualizacji. Wersję wtyczki aktualizuje człowiek, świadomie, razem z przeglądem zmian w `addons/`.

Test

Start edytora w trybie bezgłowym (`--headless --editor --quit`) kończy się bez błędów (zob. `docs/krok-zero-raport.md`).

[DO UZUPEŁNIENIA: dokładna treść komunikatu błędu sprzed poprawki.]

Zadanie

Krok zero, PR #2.

Opublikowano z repozytorium Horrorek (commit 87717ad) skryptem publikuj-bookstack.ps1.

PowerShell 5.1: tekst na stderr przerywa skrypt testów

PowerShell 5.1: tekst na stderr przerywa skrypt testów

Objaw

Skrypt uruchamiający Godota w Windows PowerShell 5.1 przerywał się z błędem, choć Godot zakończył pracę poprawnie. Wystarczało ostrzeżenie Godota na wyjściu błędów, np. `ObjectDB instances were leaked at exit`.

[DO UZUPEŁNIENIA: dokładna treść błędu PowerShella sprzed poprawki.]

Przyczyna

Przy `$ErrorActionPreference = "Stop"` Windows PowerShell 5.1 zamienia każdą linię, którą natywny program (tu Godot) wypisze na stderr, w rekord błędu, jeśli program uruchomiono operatorem `&` z przekierowaniem `2>&1`. Taki rekord przerywa skrypt, nawet gdy to tylko ostrzeżenie, a kod wyjścia programu wynosi 0.

Rozwiązanie

Godot jest uruchamiany przez `Start-Process` z osobnymi plikami na stdout i stderr (`-RedirectStandardOutput`, `-RedirectStandardError`) oraz z `-NoNewWindow -Wait -PassThru`. Kod wyjścia pochodzi z `ExitCode` zwróconego procesu, a oba pliki są potem sklejane w jeden log. Tego wzorca używają `tools/krok-zero.ps1` i `tools/run-tests.ps1`.

Błędy wykrywamy świadomie. Test jest niezaliczony, gdy kod wyjścia jest różny od 0 albo w logu jest linia `SCRIPT ERROR`, `Parse Error`, `ERROR:` lub `ERROR ` (format naszych testów).

Test

`tools/run-tests.ps1` kończy się kodem 0 przy zaliczonych testach, mimo ostrzeżeń Godota na stderr. Kontrola negatywna: tymczasowy test, który wypisuje `ERROR` i kończy się kodem 1, daje w tabeli `BŁĄD`, a cały skrypt kończy się kodem 1.

Zadanie

Krok zero (`tools/krok-zero.ps1`, PR #2) i 002 (`tools/run-tests.ps1`, PR #6).

Opublikowano z repozytorium Horrorek (commit 87717ad) skryptem publikuj-bookstack.ps1.

Esc na ekranie awansu lub zapisu zamykał też menu pod spodem

Esc na ekranie awansu lub zapisu zamykał też menu pod spodem

Objaw

Esc na ekranie awansu (`LevelUpScreen`) albo na liście slotów (`SaveSlotsScreen`) zamykał ekran, ale w logu pojawiał się błąd:

```
SCRIPT ERROR: Cannot call method 'set_input_as_handled' on a null value.
   at: LevelUpScreen._input (res://core/ui/level_up_screen.gd:86)
```

Przy okazji to samo naciśnięcie zamykało menu pod ekranem: Esc nad menu pauzy („Postać”, „Wczytaj”) wznawiało grę, a Esc nad menu medytacji („Awans”, „Zapisz”) zamykało też menu ogniska. Gracz musiał otwierać wszystko od nowa.

Przyczyna

Obsługa Esc w obu ekranach najpierw wołała `close()`, a dopiero potem `get_viewport().set_input_as_handled()`. `close()` od razu wyjmuje ekran z drzewa (`remove_child`), żeby kolejny ekran mógł dostać tę samą nazwę. Węzeł poza drzewem nie ma viewportu, więc

`get_viewport()` zwracał `null` i wywołanie się wywracało.

Zdarzenie nie zostało więc oznaczone jako obsłużone i poszło dalej, do menu pod spodem (`MeditationMenu._input`, `PauseMenu._unhandled_input`). Dotychczasowe testy wołały `close()` i `select()` bezpośrednio, a nie przez zdarzenie wejścia, więc tego nie widziały.

Rozwiązanie

- `LevelUpScreen._input` i `SaveSlotsScreen._input` najpierw oznaczają zdarzenie (`get_viewport().set_input_as_handled()`, póki węzeł jest w drzewie), a dopiero potem wołają `close()`.
- `MeditationMenu._input` ma tę samą kolejność (`accept_event()`, potem `close()`). Tam błędu nie było, bo jego `close()` używa tylko `queue_free()` i węzeł zostaje w drzewie do końca klatki. Kolejność zmieniona dla spójności.
- Przejrzane pozostałe handlery wejścia (`_input`, `_unhandled_input`): `PauseMenu`, `DevMenu`, `InteractionSensor`, `CameraControl`. Ich zamknięcie nie wyjmuje węzła z drzewa, więc `get_viewport()` po nim działa. `TargetLock` i `PlayerCombat` nie mają handlerów, tylko odpytują `Input` w klatce. Handlery przycisków ekranów („Wróć”, wybór slotu) po `close()` nie sięgają już do drzewa.

Zasada na przyszłość: **najpierw oznacz zdarzenie i weź z drzewa, czego potrzebujesz, dopiero potem zamknij albo usuń węzeł.**

Test

`tests/test_screen_input.gd` wysyła prawdziwe zdarzenie `ui_cancel` (`root.push_input`) do ekranów otwartych z menu pauzy („Postać”, „Wczytaj”) i z menu medytacji („Awans”, „Zapisz”). Sprawdza, że:

- ekran się zamyka;
- menu pod spodem zostaje otwarte, a drugi Esc zamyka dopiero je;
- w czasie testu nie ma żadnego błędu skryptu. Liczy je własny `Logger` (`OS.add_logger`), a `run-tests.ps1` dodatkowo szuka `SCRIPT ERROR` w logu.

Przed poprawką test kończył się czterema `SCRIPT ERROR`, a menu pauzy i medytacji zamykały się razem z ekranem.

Zadanie

007B-poprawka, gałąź `agent/claude/faza1-esc-ekrany`.

Opublikowano z repozytorium Horrorek (commit 87717ad) skryptem publikuj-bookstack.ps1.

Śmierć w trakcie przejścia: gracz wstawał w miejscu śmierci

Śmierć w trakcie przejścia: gracz wstawał w miejscu śmierci

Objaw

Gracz zginął w trakcie przejścia: portalem, przy zaśnięciu albo przy obudzeniu. Po ekranie śmierci wstawał z pełnym zdrowiem tam, gdzie zginął, albo w pokoju docelowym portalu, a nie przy ostatnim ognisku. Gdy śmierć przyszła w trakcie zaśnięcia, mógł też obudzić się we śnie. Zdarzało się, że przejście w ogóle się nie kończyło: `LocationManager` albo `RealmManager` zostawał „w trakcie” i portale przestawały działać.

Przyczyna

Dwie rzeczy naraz:

1. `DeathDirector.handle_death` wołał `RealmManager.respawn_now` raz i nie sprawdzał wyniku. W trakcie przejścia `respawn_now` zwraca `false`. Mimo to `restore_stats()` przywracał zdrowie, więc gracz wstawał w miejscu. Przejście kończyło się później i zmieniało pokój albo rzeczywistość już po „powrocie”. `respawn_now` nie sprawdzał też przejścia pokoju (`LocationManager`), tylko własne.

2. Śmierć i przejście używają tej samej nakładki zaciemnienia (`FadeOverlay`). Nowe zaciemnienie przerywało tween poprzedniego. Przerwany tween nie wysyła sygnału `finished`, więc `await` przejścia czekał w nieskończoność.

Rozwiązanie

- `DeathDirector` najpierw czeka co klatkę, aż skończą się przejścia w `RealmManager` i `LocationManager` (`is_transition_running()`). Po `transition_timeout` (3 s) wypisuje ostrzeżenie. Potem ponawia `respawn_now` co `respawn_retry_seconds` (0.1 s), aż się uda. Zdrowie i poczytalność wracają dopiero po udanym powrocie.
- Gdy ponawianie nie udaje się przez `respawn_give_up_seconds` (5 s), `DeathDirector` zgłasza `push_error` i wymusza powrót. Zdejmuje flagi zajętości przez nowe metody `RealmManager.force_reset()` i `LocationManager.force_reset()`, a potem woła `respawn_now` i wysyła sygnał `respawn_forced`. Ponawianie bez końca zostawiłoby gracza na czarnym ekranie śmierci.
- `RealmManager.respawn_now` zwraca `false` także w trakcie przejścia pokoju.
- `FadeOverlay`: przerwane zaciemnienie albo rozjaśnienie kończy poprzedni `await` (własny sygnał wysyłany przy przerwaniu i przy końcu tweena).

Zasada na przyszłość: **sprawdzaj wynik operacji, która może odmówić, i nie rób dalszych kroków, dopóki się nie powiedzie.**

Test

`tests/test_death.gd` (sprawdzenia 17–28):

- przerwane zaciemnienie kończy `await`;
- śmierć zaraz po rozpoczęciu przejścia portalem z A do B, zaśnięcia w A i obudzenia z B. Po każdej z nich gracz jest żywy, na jawie, w pokoju A przy ognisku, a żadne przejście nie wisi;
- w chwili, gdy zdrowie pierwszy raz wraca powyżej zera, gracz jest już na jawie przy ognisku, a nie w miejscu śmierci;
- jeden sygnał `death_started` i jeden `respawned`; druga śmierć i drugie `handle_death()` w trakcie obsługi są ignorowane;
- symulowane zawieszone przejście (flagi zajętości obu menedżerów ustawione na stałe) kończy się jednym wymuszonym powrotem: ekran śmierci schowany, zaciemnienie zdjęte, gracz żywy przy ognisku, flagi zdjęte. W teście błąd jest wypisywany zwykłym `print` (`report_forced_respawn_as_error = false`), bo `run-tests.ps1` traktuje każdą linię `ERROR:` jako błąd.

Przed poprawką test przerywał się na pierwszym z tych przypadków.

Zadanie

008B, poprawka w tym samym PR, gałąź `agent/claude/faza1-smierc`.

Opublikowano z repozytorium Horrorek (commit 87717ad) skryptem publikuj-bookstack.ps1.