Mad machines - Story Cards

TITLE	USER STORY	ACCEPTANCE CRITERIA		PRIORITY	ESTIMATION	DESCRIPTION
Story 01: Komponente platzieren	*Als Editor-Spieler möchte ich im Editiermodus per Klick eine Physik-Komponente (z. B. Platte, Ball, Domino) platzieren können, damit ich mein Maschinen-Layout zusammenbauen kann.*	1	Im Editiermodus ist eine Palette aller verfügbaren Objekte sichtbar.	HIGH	**3 story points**	Basisfunktion für alle Editier-Workflows.
		2	Durch Klick auf ein Objekt in der Palette wechsle ich in den „Platzieren"-Modus.			
		3	Ein Klick auf die Spiel-Canvas fügt das gewählte Objekt an dieser Position ein.			
		4	Das Objekt wird unmittelbar angezeigt und in der Level-Datenstruktur registriert.			
		5	Ein erneutes Speichern/Exportieren erzeugt eine JSON-Datei mit dem neuen Objekt.			
Story 02: Komponente entfernen	*Als Editor-Spieler möchte ich bereits platzierte Objekte per Klick entfernen können, um Fehler zu korrigieren.*	1	Im Editiermodus kann ich in den „Löschen"-Modus wechseln.	HIGH	**3 story points**	Ergänzung zur Platzier-Funktion, verbessert Usability.
		2	Ein Klick auf ein bestehendes Objekt löscht es aus Canvas und Level-Daten.			
		3	Nach Speichern/Export ist das Objekt nicht mehr in der Level-Datei vorhanden.			
		4	Undo/Redo kann diesen Schritt rückgängig machen.			
		5				
Story 03: Simulation starten / stoppen	*Als Spieler möchte ich nach dem Editieren per Klick auf „Play" die Physik-Simulation starten und mit „Stop" abbrechen können, um mein Konstrukt zu testen.*	1	Ein „Play/Stop"-Button wechselt zwischen Editor- und Simulationsmodus.	HIGH	**5 story points**	Herzstück des Spielerlebnisses.
		2	Im Simulationsmodus wird jBox2D-Physik aktiv und Objekte reagieren auf Kräfte.			
		3	Ein Klick auf „Stop" pausiert die Simulation und wechselt zurück in den Editiermodus.			
		4	Der Editor-Zustand vor der Simulation bleibt erhalten (Positionen, Objekte).			
		5	Ein Reset-Button stellt den ursprünglichen Editierzustand wieder her.			
Story 04: Levels speichern & laden	*Als Entwickler möchte ich Level per JSON exportieren und wieder importieren können, damit sie teilbar und versionierbar sind.*	1	Im Menü existieren „Speichern" und „Laden"-Einträge.	MEDIUM	**3 story points**	Grundlage für Level-Sharing.
		2	„Speichern" erzeugt eine leicht lesbare JSON-Datei im levels/-Verzeichnis.			
		3	„Laden" zeigt eine Liste vorhandener Level an und lädt die gewählte Datei.			
		4	Canvas und Editor-Status entsprechen nach dem Laden exakt der JSON-Definition.			
		5	Fehlermeldung, falls Datei fehlt oder ungültig ist.			
Story 05: Puzzle-Modus & Level-Progression	*Als Puzzle-Spieler möchte ich eine Abfolge vordefinierter Level mit begrenzter Objekt-Auswahl spielen können, um gestellte Aufgaben zu lösen.*	1	Nach Start von Puzzle-Modus sehe ich ein Level-Auswahlmenü.	HIGH	**8 story points**	Kern-Spielmodus mit Herausforderung und Lernkurve.
		2	Jedes Level hat ein Inventar mit begrenzten Bauteilen.			
		3	Siegesbedingung (z. B. Ball im Ziel) wird angezeigt und geprüft.			
		4	Nach erfolgreichem Lösen wechsle ich automatisch zum nächsten Level.			
		5	Fehlversuch erlaubt unbegrenzte Wiederholungen.			
Story 06: Inventarsystem & Einschränkungszonen	*Als Level-Designer möchte ich Bauteile-Limits pro Level festlegen und Platzierungszonen definieren, um Gameplay-Restriktionen zu steuern.*	1	Level-JSON enthält inventory mit Stückzahlen pro Objekttyp.	MEDIUM	**5 story points**	Steuert Spiellogik und Schwierigkeitskurve.
		2	Editiermodus zeigt Restbestand in Echtzeit an.			
		3	Platzierungsverbotszonen werden als halbtransparente Flächen angezeigt.			
		4	Versuche, in Plazierungsverbotszonen zu platzieren, werden blockiert und gemeldet.			
		5	UI gibt Warnung bei Erreichen der Inventar-Grenze.			
Story 07: Hauptmenü & Navigation	*Als Spieler möchte ich ein Hauptmenü mit den Optionen „Puzzle-Modus", „Sandbox", „Einstellungen" und „Beenden" haben, damit ich schnell starten kann.*	1	Hauptmenü erscheint beim Spielstart.	MEDIUM	**5 story points**	Wahlfeature für bessere UX, Alt+F4 Vermeidung.
		2	Auswahl „Puzzle-Modus" öffnet Level-Auswahl.			
		3	„Sandbox" wechselt direkt in Editier-Modus ohne Limits.			
		4	„Einstellungen" öffnet Popup für Lautstärke und Texturpaket.			
		5	„Beenden" schließt das Programm sauber.			
Story 08: Level-Auswahlmenü	*Als Spieler möchte ich im Puzzle-Modus alle verfügbaren Level mit Miniaturansichten und Kurzbeschreibungen sehen, um gezielt auszuwählen.*	1	Menü listet Level als Karten mit Thumbnail und Titel.	MEDIUM	**5 story points**	Verliefft Menüs, erhöht Übersicht.
		2	Mouse-Hover zeigt kurze Level-Description.			
		3	Klick auf Karte lädt und startet Level.			
		4	Gelöste Level sind visuell markiert.			
		5				
Story 09: Drag & Drop & Rotate im Editor	*Als Editor-Spieler möchte ich platzierte Objekte per Drag & Drop verschieben und per Drehrad oder Taste rotieren können, um präzises Layout zu gestalten.*	1	Objekt anklicken → Drag-Modus, Ziehen bewegt es auf Canvas.	MEDIUM	**8 story points**	Erweiterter Editor-Komfort (Level-Editor+).
		2	Mit Mausrad oder UI-Button lässt sich die Ausrichtung um 15°-Schritte drehen.			
		3	Echtzeit-Vorschau während Drag/Rotate.			
		4	Änderungen werden in Level-Daten sofort aktualisiert.			
		5	Undo/Redo unterstützt diese Aktionen.			
Story 10: Undo/Redo im Editor	*Als Editor-Spieler möchte ich letzte Editier-Schritte rückgängig bzw. wiederherstellen können, um Fehler schnell zu korrigieren.*	1	Zwei Buttons „Undo" & „Redo" sind in der Toolbar vorhanden.	LOW	**5 story points**	Erhöht Usability.
		2	Jeder Platzieren-/Löschen-/Rotate-/Drag-Schritt lässt sich einzeln rückgängig machen.			
		3	Bis zu 10 letzte Aktionen werden gespeichert.			
		4	Undo/Redo ändert Canvas und Level-Daten konsistent.			
		5				
Story 11: Maschinen-Interface	*Als Entwickler möchte ich das Spiel über Kommandozeile ohne GUI starten und Protokolle von Simulationsergebnissen als Text erhalten, um automatisierte Tests oder KI-Agenten-Training zu ermöglichen.*	1	Start mit –no-gui lädt Level und führt Simulation durch.	LOW	**8 story points**	Erlaubt KI-Integration.
		2	Konsolenausgabe zeigt Positionen, Kollisionen und Endzustand.			
		3	Exit-Code = 0 bei erfolgreicher Simulation, ≠ 0 bei Fehlern.			
		4	Protokoll im JSON- oder CSV-Format optionierbar.			
		5	Dokumentation der CLI-Optionen im Handbuch.			
Story 12: CI-Pipeline & Automatisierte Tests	*Als Entwickler möchte ich eine GitLab-CI einrichten, die bei jedem Push kompiliert, testet (JUnit, JaCoCo), analysiert (PMD) und einen Bericht veröffentlicht, um Code-Qualität sicherzustellen.*	1	.gitlab-ci.yml definiert Stages: build, test, report.	MEDIUM	**3 story points**	Non-functional, sichert Stabilität im Team.
		2	Maven-Befehle mvn clean compile test site laufen fehlerfrei durch.			
		3	Artefakte (Site-Reports) werden als CI-Job-Artefakte veröffentlicht.			
		4	Bei Test-Failure schlägt Pipeline fehl.			
		5	Coverage-Badge und Build-Status-Badge im README verlinkt.			