Testdokument

Projektname: ProductivityGarden

Name	Jonas Huber
Matrikelnummer	IU14085128
Modul	Projekt: Software Engineering DLMCSPSE01_D
Datum	24.01.2025

Inhalt

Abbildungsverzeichnis

1. Teststrategie

Die Teststrategie beschreibt die Maßnahmen zur Qualitätssicherung der Anwendung ProductivityGarden. Ziel ist es, sicherzustellen, dass die Anwendung fehlerfrei, sicher, performant und benutzerfreundlich ist.

Whitebox-Tests

Bei Whitebox-Tests ist die interne Struktur und Funktionsweise des Codes bekannt und zugänglich, sodass gezielt einzelne Komponenten und deren Interaktionen auf Korrektheit und Effizienz geprüft werden können. Um die Qualität in sämtlichen Entwicklungsphasen gewährleisten zu können, erfolgt eine Unterteilung in drei Teststufen.

Unit-Tests

Unit-Tests konzentrieren sich auf die kleinsten Testeinheiten des Codes, meist Funktionen oder Methoden. Das übergeordnete Ziel besteht in der Sicherstellung der korrekten Funktionsweise einzelner Code-Einheiten, unabhängig von anderen Einheiten.

Diese Art von Tests können somit direkt nach der Fertigstellung einzelner Code-Einheiten durchgeführt werden, ohne dass weitere Abhängigkeiten berücksichtigt werden müssen.

Unit-Tests umfassen die Prüfung der Funktionalität, beispielsweise von Timer-Funktionalitäten wie des Pomodoro-Timers, des anpassbaren Timers und der Stoppuhr. Des Weiteren werden durch Unit-Tests die Korrektheit beim Speichern und Laden von Daten sowie bei Benutzerinteraktionen (Ereignisse wie Button-Klicks) überprüft.

Testmethoden:

- Manuelle Tests
- Automatisierte Unittests:
  Verwendung von Tools wie *unittest* für die Testautomatisierung

Integrationstests

Integrationstests überprüfen, ob die Interaktion zwischen verschiedenen Code-Einheiten reibungslos funktioniert.

Dabei wird beispielsweise überprüft, ob produktive Zeit korrekt in Punkte umgewandelt wird und einem Projekt zugeordnet wird.

Folglich müssen sämtliche Code-Einheiten, die mit den Tests in Zusammenhang stehen, vorab fertiggestellt werden.

Arten von Integrationstests:

- Big-Bang-Ansatz:
  Alle Module werden gleichzeitig integriert und getestet.
- Inkrementeller Ansatz:
  Module werden schrittweise integriert und getestet. (Top-Down-Ansatz oder Bottom-Up-Ansatz)

- Funktionale Integration:
  Testet Funktionen, die mehrere Module betreffen.

Testmethoden:

- Manuelle Tests
- Optional:
  - Testframeworks wie *pytest* mit Abdeckungsberichten (*pytest-cov*)
  - Integrationstools wie *tox*, um unterschiedliche Konfigurationen zu testen
  - Verwendung von Tools wie *pytest-qt*, *Selenium* oder *PyAutoGUI* für automatisierte GUI-Tests bzw. zur Automatisierung von Benutzerinteraktionen.

Systemtests
Im Rahmen von Systemtests erfolgt eine Prüfung der Anwendung als Ganzes unter realistischen Bedingungen. Dabei ist sicherzustellen, dass die Anwendung wie vorgesehen funktioniert und den Anforderungen entspricht.

Abdeckung:

- Benutzerschnittstelle
- Performance
  - Reaktionsgeschwindigkeit
  - Flüssigkeit der Animationen
  - Auslastung von Hardware-Ressourcen (wie CPU, GPU, RAM, ROM)
- Hintergrundbetrieb
- Kompatibilität

Testmethoden:

- Manuelle Tests:
  - Usability-Tests durch Benutzer, um die Benutzerfreundlichkeit und Performance zu bewerten.
  - Ressourcen-Auslastung mit Hilfe von Windows Task-Manager überwachen
- Optional:
  - Verwendung von Tools wie *pytest-qt*, *Selenium* oder *PyAutoGUI* für automatisierte GUI-Tests bzw. zur Automatisierung von Benutzerinteraktionen.

Weitere Tests
Zur weiteren Abdeckung relevanter Aspekte der Qualitätssicherung werden Statische Tests und bei Möglichkeit auch Blackbox-Tests herangezogen.

Statische Tests
Im Rahmen statischer Tests erfolgt keine Ausführung der zu untersuchenden Software. Im Rahmen eines sogenannten Code-Review wird die Software einer analytischen Überprüfung der einzelnen Code-Zeilen unterzogen, um deren Sinnhaftigkeit zu evaluieren. So kann beispielsweise eine Datenflussanalyse Aufschluss darüber geben, welche Daten herangezogen und weitergegeben werden und ob dies den Anforderungen entspricht.

Sinnvoll ist in diesem Zusammenhang der Einsatz eines Tools, welches den Quellcode mit Hilfe von konfigurierbaren Regeln automatisiert durchdringt und das Ergebnis in einer übersichtlichen Form ausgibt. Beispiele hierfür sind *Pylint*, *Flake8* & *Mypy*.

Blackbox-Tests
Im Gegensatz zu Whitebox-Tests ist dem Tester bei Blackbox-Tests der Code nicht bekannt. Der Fokus liegt hierbei auf der Überprüfung des sichtbaren Ergebnisses. Der Vorteil von Blackbox-Tests liegt in der Betrachtung der Software aus der Perspektive des Endnutzers.

Vorgehen zur Testdurchführung

Testplanung

- Definieren der Testfälle basierend auf den funktionalen und nicht-funktionalen Anforderungen in der Testspezifikation
    - Testfall mit eindeutiger ID
    - Testziel des Testfalls
    - Voraussetzungen die in der Anwendung vor der Ausführung des Tests hergestellt werden müssen
    - Eingabedaten und/oder auszuführende Aktionen der Benutzer:innen zur Durchführung des Tests
    - Erwartetes Ergebnis
- Priorisierung der Tests

Testdurchführung

- Whitebox-Tests
    - Unit-Tests während der Entwicklungsphase durchführen
    - Integrationstests ebenfalls während der Entwicklungsphase, aber jeweils nach Implementierung einzelner Module durchführen
        - Anfänglich wird mit dem inkrementellen Ansatz (Bottom-Up) versucht die Integrationstests abzudecken.
        - In speziellen Fällen können im Nachhinein auch noch weitere Tests mit dem Funktionalen Integrationstest-Ansatz notwendig sein.
    - Systemtests in der finalen Testphase vor der Veröffentlichung.
- Weitere Tests
    - Statische Tests während der Entwicklungsphase durchführen
    - Blackbox-Tests nach Möglichkeit in der finalen Testphase vor der Veröffentlichung (zusammen mit Systemtests)

Testprotokoll

- Erstellung des Testprotokolls im Anschluss an die Durchführung der Tests.
    - Umfasst die Testfälle aus der Testspezifikation sowie die tatsächlichen Ergebnisse

Testpriorisierung

Abbildung 1 zeigt die Testpriorisierung in drei Phasen aufgeteilt, basierend auf der Teststrategie und der Wichtigkeit der jeweiligen Module. Die Tests werden nach ihrer Kritikalität für die Anwendung priorisiert. Diese Priorisierung gewährleistet eine schrittweise Sicherstellung der Funktionalität, von den kleinsten Einheiten bis hin zum gesamten System. Zusätzlich wird sichergestellt, dass Fehler frühzeitig erkannt werden.

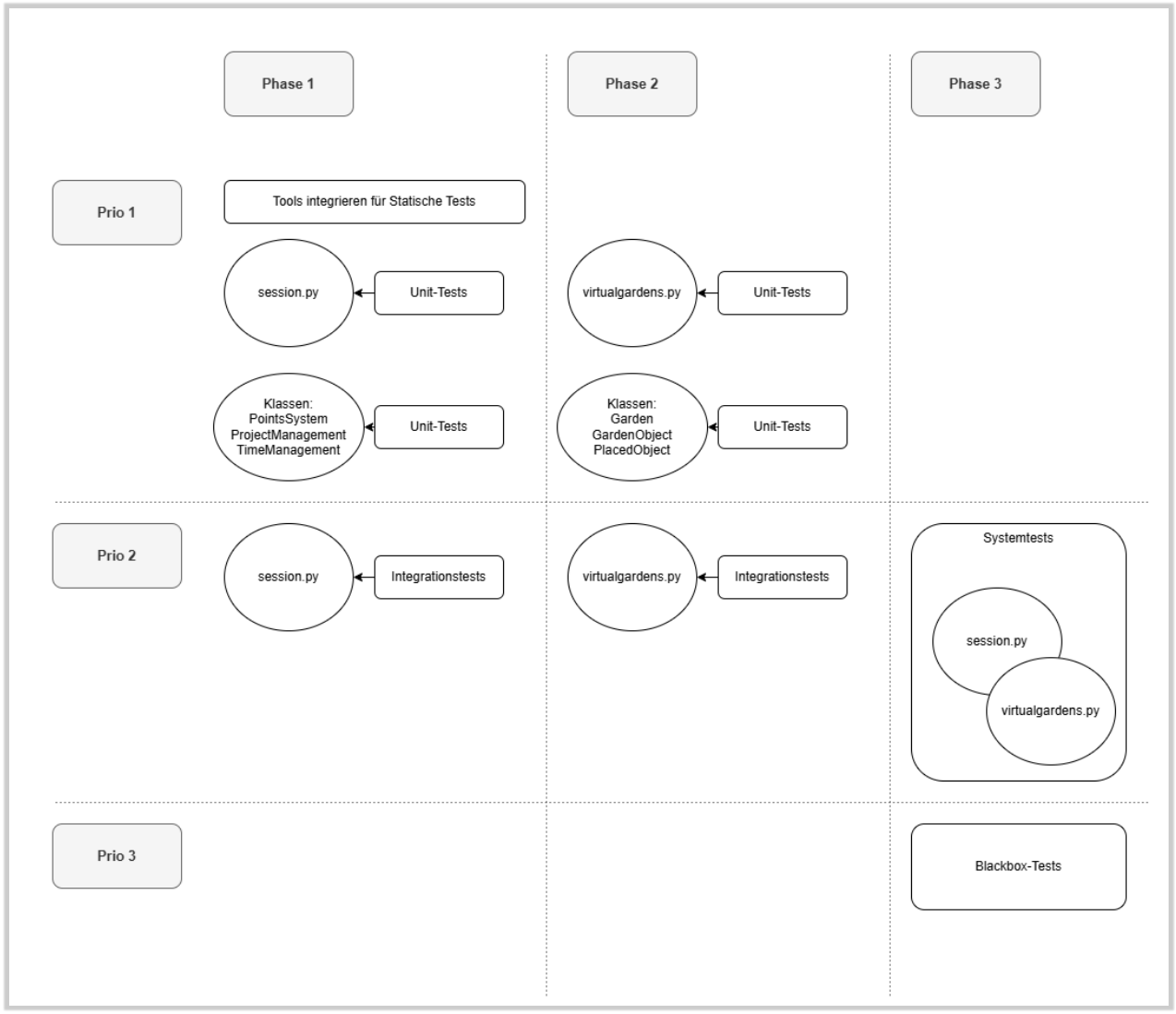

Abbildung 1 – Diagramm Testpriorisierung

2. Testprotokoll

Whitebox-Tests

Unit-Tests

Die Auflistung aller Unit-Tests würde den Rahmen dieses Dokuments sprengen und diese sind in den Dateien „..._test.py“ im Ordner „unittests“ ersichtlich. Aus diesem Grund werden die Tests der Komponenten „session.py“ hier nur zu Demonstrationszwecken aufgeführt:

Testfall 1: Initialisierung der Benutzeroberfläche

Testfall-ID: UTP01

Testziel:
Überprüfung der initialen Konfiguration der Benutzeroberfläche.

Voraussetzungen:
Die Anwendung wird gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
keine	1. Titel des Fensters: "ProductivityGarden" 2. Fensterbreite entspricht „WIDTH“ 3. Fensterhöhe entspricht „HEIGHT“	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 2: Initialisierung der Punkteübersicht

Testfall-ID: UTP02

Testziel:

Validierung der initialen Punktanzahl in der Benutzeroberfläche.

Voraussetzungen:

- Ein Testdatensatz mit Punktinformationen ist in der Datenbank vorhanden.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
keine	Die Punkte in circle_av und circle_tot entsprechen den Werten der Punktverwaltung	

Testfall 3: Initialisierung des Timer-Modus

Testfall-ID: UTP03

Testziel: Überprüfung des initialen Zustands des Timer-Modus.

Voraussetzungen:

- Die Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
keine	1. ausgewählter Timer-Modus ist „pomodoro“ 2. Zeitanzeige: „00:00:00“	

Testfall 4: Validierung der Timer-Eingabe

Testfall-ID: UTP04

Testziel: Überprüfung der Validierung von Benutzereingaben für den Timer.

Voraussetzungen:

- Die Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. "01:00:00" 2. "25:00:00" 3. "00:00:59" 4. "invalid"	1. True 2. False 3. False 4. False	1. True 2. False 3. False 4. False

Testfall 5: Start des Timers

Testfall-ID: UTP05

Testziel: Überprüfung der Funktionalität des Timer-Starts.

Voraussetzungen:

- Die Eingaben für Arbeitszeit und Pausenzeit wurden gesetzt.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setzen von pomodoro_work_input auf "00:25:00". 2. Setzen von pomodoro_break_input auf "00:05:00". 3. Ausführung von handle_start_time().	Timer-Modus wechselt zu: „running“	erfüllt

Testfall 6: Pausieren des Timers

Testfall-ID: UTP06

Testziel: Überprüfung der Funktionalität des Timer-Pausierens.

Voraussetzungen:

- Der Timer läuft.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Ausführung von handle_pause_time() -> 2x	1. Der Modus wechselt zu "paused". 2. Der Modus wechselt zurück zu "running".	1. erfüllt 2. erfüllt

Testfall 7: Stoppen des Timers

Testfall-ID: UTP07

Testziel: Überprüfung der Funktionalität des Timer-Stoppens.

Voraussetzungen:

- Der Timer läuft.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Ausführung von handle_stop_time()	Der Modus wechselt zu „stopped“	erfüllt

Testfall 8: Umschalten des Timer-Modus

Testfall-ID: UTP08

Testziel: Validierung der Funktionalität des Umschaltens zwischen verschiedenen Timer-Modi.

Voraussetzungen:

- Die Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Ausführung von handle_toggle_mode(). -> 3x	Modus wechselt zu 1. „timer“ 2. „stopwatch“ 3. „pomodoro“	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 9: Speichern und Laden von Daten

Testfall-ID: UTP09

Testziel: Überprüfung des Speicherns und Ladens von Benutzerdaten.

Voraussetzungen:

- Daten wurden in die Felder eingegeben.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setzen von Punkten und Texteingaben. 2. Ausführung von save_json_data(). 3. Start einer neuen Sitzung und Ausführung von load_json_data().	Die gespeicherten Werte werden korrekt geladen.	erfüllt

Integrationstests

Dieser Abschnitt beinhaltet die verschiedenen Integrationstests, die manuell durchgeführt wurden. Die in der Teststrategie als optional deklarierten Testmethoden (siehe Integrationstests S.2) wurden aus zeitlichen Gründen nicht implementiert, jedoch würden diese sich für eine umfassendere Testabdeckung zukünftig lohnen. Die Testfälle sind wieder aufgeteilt in die zwei Hauptkomponenten Produktivanwendung und Spielkomponente

Allgemeine Voraussetzungen / Testumgebung:

- Das Python-Projekt ist lokal installiert/ausführbar.
- Standard-Einstellungen in der constants.py sind valide (z.B. Pfade, Farbcodes).
- Benötigte Python-Pakete (PyQt6, PyQt6.QtCharts etc.) sind installiert.

Für Produktivanwendung (session.py):

Testfall 1: Anwendung starten mit leerem/fehlendem JSON

Testfall-ID: ITP01

Testziel: Überprüfen, dass beim ersten Start oder bei fehlender JSON-Datei die Default-Werte korrekt initialisiert werden.

Voraussetzungen:

- Die JSON-Datei (JSON_FILE) existiert nicht oder ist leer.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Sicherstellen, dass die JSON-Datei entweder gelöscht oder geleert wurde. 2. Starte die Anwendung (MainSession).	1. Die Anwendung erzeugt eine neue JSON-Datei mit Default-Werten (z.B. total_points = 0, available_points = 0, voreingestellte Pomodoro-Zeiten etc.). 2. Keine Fehler oder Fehlermeldungen in der GUI. 3. Im GUI werden 0 Punkte (total/available) angezeigt und die voreingestellten Felder für Timer/Pomodoro sind sichtbar (z.B. "00:25:00" / "00:05:00").	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 2: Start/Pause/Stop der Stoppuhr (Stopwatch)

Testfall-ID: ITP02

Testziel: Verifizieren, dass der Stopwatch-Modus korrekt funktioniert und die GUI-Elemente (Zeitlabel, Buttons) richtig reagieren.

Voraussetzungen:

- Anwendung ist gestartet.
- Der Timer-Modus steht auf „Stopwatch“ (ggf. umschalten über den „Switch to ...“-Button).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke auf den Start-Button. 2. Beobachte das Zeitlabel (z.B. clock_label) für einige Sekunden. 3. Klicke auf Pause. 4. Warte einige Sekunden und überprüfe, ob die Zeit weiterhin eingefroren bleibt. 5. Klicke auf Resume. 6. Beobachte das Zeitlabel wieder einige Sekunden. 7. Klicke auf Stop.	1. Nach Schritt 1 läuft die Stoppuhr los (z.B. 00:00:01, 00:00:02, ...). 2. Nach Schritt 3 wechselt der Button-Text auf „Resume“ und die Uhrzeit friert ein. 3. Nach Schritt 5 zählt die Stoppuhr an der eingefrorenen Stelle weiter (keine Nullstellung). 4. Nach Schritt 7 wird die Zeit auf 00:00:00 zurückgesetzt, und der Button-Text ändert sich zu „Pause“.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 3: Umschalten zwischen Pomodoro, Timer und Stopwatch

Testfall-ID: ITP03

Testziel: Validieren, dass das Umschalten des Timer-Modus per Button die Eingabefelder und Labels korrekt anpasst.

Voraussetzungen:

- Anwendung ist gestartet und befindet sich in irgendeinem Modus.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Stelle sicher, dass der aktuelle Modus z.B. „Pomodoro“ ist. 2. Klicke auf den Button „Switch to Timer“. 3. Beobachte die GUI: Pomodoro-Felder (Work/Break) sollten verschwinden, das Timer-Eingabefeld erscheint. 4. Klicke erneut auf den Button, bis „Stopwatch“ erscheint. 5. Beobachte, dass nun weder Pomodoro- noch Timer-Eingabefelder zu sehen sind.	1. Bei „Pomodoro“ sind zwei Eingabefelder sichtbar (pomodoro_work_input, pomodoro_break_input) und korrekt beschriftet. 2. Bei „Timer“ wird nur das Timer-Eingabefeld (timer_input_field) angezeigt, während Pomodoro-Felder ausgeblendet sind. 3. Bei „Stopwatch“ sind alle Eingabefelder ausgeblendet. 4. Die Label timer_mode_label zeigt die richtige Beschriftung (z.B. „POMODORO“, „TIMER“, „STOPWATCH“).	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 4: Pomodoro-Modus starten mit validen und invaliden Zeiten

Testfall-ID: ITP04

Testziel: Überprüfen, dass Pomodoro-Work/Break-Einstellungen korrekt validiert werden und der Ablauf stimmt.

Voraussetzungen:

- Anwendung ist gestartet, Modus ist auf „Pomodoro“ eingestellt.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setze pomodoro_work_input auf "00:25:00". 2. Setze pomodoro_break_input auf "00:05:00". 3. Klicke auf Start. 4. Prüfe, ob das Label die verbleibende Zeit anzeigt („Work: 00:25:00“, läuft herunter). -- Erneut mit invalide Eingabe -- 1. Trage in pomodoro_work_input einen ungültigen Wert ein, z.B. "25:00" (falsches Format ohne Stunden/Minuten-Trennung) oder "00:00:30" (unter 1 Minute). 2. Klicke auf Start.	1. Keine Fehlermeldung erscheint. 2. Die Pomodoro-Uhr beginnt im Work-Abschnitt zu laufen. 3. Nach Ablauf der Work-Zeit (falls gewartet wird) schaltet es automatisch in den Break-Modus um. -- invalide Eingabe: -- 1. Eine Fehlermeldung in roter Farbe (input_error_label) weist auf ein falsches Format oder auf die Mindestzeit hin. 2. Die Pomodoro-Uhr startet nicht.	1. erfüllt 2. erfüllt 3. erfüllt -- invalide Eingabe: -- 1. erfüllt 2. erfüllt

Testfall 5: Timer-Modus starten, ablaufen lassen und prüfen^

Testfall-ID: ITP05

Testziel: Verifizieren, dass der Timer-Modus die Zeit korrekt herunterzählt und bei Ablauf stoppt.

Voraussetzungen:

- Anwendung ist gestartet, Modus ist ggf. auf „Timer“ umgeschaltet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setze das Feld timer_input_field auf "00:00:05" (5 Sekunden zum Testen). 2. Klicke auf Start. 3. Beobachte das clock_label. 4. Warte, bis die 5 Sekunden abgelaufen sind.	1. Die Uhr zählt rückwärts von 00:00:05 auf 00:00:00. 2. Bei 00:00:00 stoppt die Uhr. 3. Modus wechselt in den Zustand „stopped“ (Button „Stop“ hat keinen Effekt mehr bzw. ist wieder auf Default).	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 6: Anlegen eines neuen Projekts und Aktualisierung der Übersicht

Testfall-ID: ITP06

Testziel: Sicherstellen, dass beim Hinzufügen eines Projekts (handle_add_new_project) die Datenbankeinträge und GUI-Elemente (Dropdown, Infofelder) aktualisiert werden.

Voraussetzungen:

- Anwendung ist gestartet.
- Es existiert mindestens 1 Projekt in der Datenbank.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke im Project-Bereich auf Add. 2. Beobachte das Dropdown projects_dropdown. 3. Ein neues Projekt mit Standardwerten (z.B. "New Project" o. Ä.) sollte ausgewählt sein. 4. Ändere den Namen im Feld pr_name_input auf z.B. "Testprojekt" und beobachte, ob sich der Eintrag im Dropdown direkt ändert. 5. Klicke z.B. in ein anderes Feld oder warte, bis der Datensatz gespeichert wird.	1. Dropdown enthält einen neuen Eintrag. 2. Die Anzeige in pr_name_input und der Dropdown-Eintrag stimmen überein. 3. Kein Fehler im Log / keine Exception. 4. Bei Aufruf von ProjectManagement.get_projects_name_list(...) taucht das neue Projekt in der Rückgabe auf.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 7: Löschen eines Projekts

Testfall-ID: ITP07

Testziel: Prüfen, dass das Löschen des aktuell gewählten Projekts korrekt die Datenbank und GUI anpasst.

Voraussetzungen:

- Es sind mindestens 2 Projekte in der Datenbank (damit das Löschen bemerkbar ist).
- Ein Projekt ist im Dropdown angewählt.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Stelle sicher, dass im Dropdown das zu löschende Projekt ausgewählt ist. 2. Klicke auf Delete. 3. Beobachte das Dropdown.	1. Das gelöschte Projekt verschwindet aus dem Dropdown. 2. Die Anwendung schaltet (automatisch) auf das nächste verfügbare Projekt um (z.B. Index 0). 3. Kein Fehler/Absturz. 4. ProjectManagement.get_id_by_name(...) sollte für das gelöschte Projekt nun None oder Fehler liefern, weil es nicht mehr existiert.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 8: Manuelles Hinzufügen von Zeit zu einem Projekt

Testfall-ID: ITP08

Testziel: Überprüfen, dass die Eingabe im Feld pr_add_time korrekt validiert wird und die Projektzeit entsprechend hochgezählt wird.

Voraussetzungen:

- Ein bestehendes Projekt ist ausgewählt.
- Zeitmanagement kann gestoppt sein (zur Vereinfachung).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Trage im Feld pr_add_time einen gültigen Integer ein, z.B. "15". 2. Klicke auf den Button Add time. 3. Beobachte die Anzeige (Kreis circle_project_time). 4. Wiederhole den Vorgang mit einem ungültigen Wert, z.B. "abc" oder "1000".	1. Bei "15" wird die Projektzeit (in Minuten) um 15 erhöht. circle_project_time zeigt entsprechend einen um 15 erhöhten Wert. 2. Bei Eingaben außerhalb des erlaubten Bereichs (1–999) oder nicht numerischen Werten erscheint eine Fehlermeldung im GUI (gui_show_error). 3. Keine Exceptions im Log	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 9: Punkte-Update alle 10 produktiven Minuten

Testfall-ID: ITp09

Testziel: Sicherstellen, dass alle 10 gesammelten Produktiv-Minuten (z.B. im Stopwatch-Modus) automatisch Punkte gutgeschrieben werden.

Voraussetzungen:

- Es existiert mindestens 1 Projekt. (Wird automatisch angelegt beim erstmaligen Starten der Anwendung)
- Punktesystem steht zu Beginn auf einem bekannten Wert (z.B. total=0, available=0).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Starte die Stoppuhr (oder füge manuell Zeit hinzu, sodass self.time_manager.productiv_minutes ansteigt). 2. Erzeuge mindestens 10 produktive Minuten. 3. Achte darauf, dass der interne Zähler (in minute_counter) alle 10 Minuten 1 Punkt vergibt.	1. Nach (akkumulierten) 10 produktiven Minuten erhöht sich circle_av und circle_tot beide um 1. 2. GUI zeigt diesen neuen Punktestand an. 3. Bei z.B. 20 Minuten steigt der Wert erneut um 1 Punkt.	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 10: Aktualisierung der Projekt-Pie-Chart

Testfall-ID: ITP10

Testziel: Überprüfen, dass das Pie-Chart (ProjectsOverviewPieChart) nach Änderungen an den Projektdaten (Zeit) korrekt aktualisiert wird.

Voraussetzungen:

- Mindestens 2 Projekte existieren.
- Die Projekt-Zeitdaten im Dropdown sind verschieden (z.B. Projekt A = 10 min, Projekt B = 30 min).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Wähle „Projekt A“ aus und füge manuell 10 min hinzu. 2. Lasse einige Sekunden verstreichen, damit der update_low_frequency()-Timer auslöst. 3. Beobachte das Pie-Chart. 4. Schalte auf „Projekt B“ und füge dort ebenfalls Zeit hinzu. 5. Beobachte erneut das Pie-Chart nach einigen Sekunden.	1. Die Tortenstücke in der Chart spiegeln sofort oder nach dem 2-Sekunden-Intervall die neuen Zeitwerte wider. 2. Kein Darstellungsfehler oder GUI-Absturz.	1. erfüllt 2. erfüllt

Testfall 11: Ungültige Eingabe im Projekt-Namen (Sonderzeichen, Leerzeichen)

Testfall-ID: ITP11

Testziel: Sicherstellen, dass die Eingabefelder für den Projektnamen und andere Felder nur erwartete Zeichen zulassen und ggf. Validierungen/Begrenzungen greifen.

Voraussetzungen:

- Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke auf Add, um ein neues Projekt hinzuzufügen. 2. Gib im Feld pr_name_input eine Zeichenkette mit Sonderzeichen ein, z. B. "@@@###!!!" oder eine sehr lange Zeichenkette mit über 40 Zeichen. 3. Beobachte, ob und wie das GUI reagiert (z. B. rote Rahmen, Fehlermeldung, automatisches Kürzen).	1. Das Feld darf maximal 40 Zeichen übernehmen und sollte automatisch kürzen oder eine Fehlermeldung ausgeben. 2. Sonderzeichen könnten erlaubt sein oder vom System abgewiesen werden – je nach Designvorgabe. 3. Keine Abstürze oder Exceptions.	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 12: Datenpersistenz beim Neustart (JSON)

Testfall-ID: ITP12

Testziel: Verifizieren, dass User-Eingaben (Punktestand, Timer-Einstellungen, Text aus text_box) korrekt in der JSON-Datei gespeichert und beim Neustart wiederhergestellt werden.

Voraussetzungen:

- Anwendung ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Setze in der Anwendung einige Werte: total_points = 5, available_points = 3 (durch Zeitmessung oder manuelles Hinzufügen). 2. Pomodoro-Felder auf z. B. 00:20:00 und 00:10:00. 3. Tippe einen Text in text_box. 4. Schließe die Anwendung sauber (Fenster schließen). 5. Starte die Anwendung erneut.	1. Die zuvor gesetzten Werte werden aus der JSON-Datei geladen und in der GUI angezeigt. 2. Punktestand, Timer-Eingaben und Text bleiben erhalten. 3. Keine Fehlermeldung aufgrund ungültiger oder fehlender JSON-Daten.	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 13: Projektwechsel während laufender Zeitmessung

Testfall-ID: ITP13

Testziel: Überprüfen, dass ein Wechsel des ausgewählten Projekts keine Probleme verursacht, wenn die Stoppuhr oder Timer noch läuft.

Voraussetzungen:

- Zwei verschiedene Projekte sind angelegt.
- Zeitmessung (z. B. Stopwatch) ist gestartet.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Während die Stoppuhr läuft, wähle über das Dropdown ein anderes Projekt aus. 2. Beobachte, ob die gestoppte Zeit später korrekt auf das nun selektierte Projekt gebucht wird. 3. Stoppe die Zeitmessung. 4. Prüfe den Zeitwert in beiden Projekten (via GUI-Kreise oder manuelle SELECT-Abfrage in der DB), ob eine doppelte oder falsche Zeitbuchung stattfand.	1. Die bis zum Projektwechsel gesammelten Minuten werden dem ursprünglichen Projekt zugerechnet. 2. Die gesammelten Minuten nach dem Projektwechsel, werden dem neu selektierten Projekt zugerechnet. 3. Kein Absturz, keine doppelte oder falsche Zeitbuchung.	1. erfüllt 2. erfüllt 3. erfüllt

Für Spielkomponente (virtualgardens.py):

Testfall 14: Start des Spiels & Metadaten-Cleanup

Testfall-ID: ITVG01

Testziel: Prüfen, ob das Spiel in das Hauptmenü gelangt, sowie sicherstellen, dass vorhandene .map-Dateien in gardens_data.json eingetragen sind und nicht mehr existierende Dateien entfernt werden.

Voraussetzungen:

- gardens_data.json existiert (ggf. mit Einträgen, die nicht mehr zu .map-Dateien passen).
- Mindestens eine .map-Datei im gardens Ordner die nicht in der gardens_data.json aufgeführt ist
- Mindestens eine .map-Datei die in der gardens_data.json existiert löschen

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Starte das Python-Skript (virtualgardens.py). 2. Beobachte die Konsolenausgabe zur „Cleanup“-Funktion. 3. Warte, bis das Hauptmenü erscheint.	1. Nicht mehr existierende .map-Dateien werden aus gardens_data.json entfernt (Konsolenausgabe: "[Cleanup] Removed metadata entry ..."). 2. Neue .map-Dateien, die noch nicht im JSON vorhanden sind, werden hinzugefügt (Konsolenausgabe: "[Cleanup] Added metadata entry ...").	1. erfüllt 2. erfüllt

Testfall 15: Neues Gartenprojekt anlegen (Create Garden)

Testfall-ID: ITVG02

Testziel: Integrationstest für das Anlegen eines neuen Gartens über das Hauptmenü:

- Menüauswahl („Create Garden“).
- Vegetationsauswahl über choose_vegetation().
- Namenseingabe über text_input_dialog().
- Anlage der entsprechenden GardenObjects über create_garden_objects() und Speichern in .map-Datei.

Voraussetzungen:

- Spiel befindet sich im Hauptmenü
- gardens_data.json ist vorhanden/synchronisiert.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke im Hauptmenü auf „Create Garden“. 2. Wähle im darauf folgenden Vegetationsdialog „City Park“ (oder eine andere Option) per Mausklick. 3. Im Namenseingabedialog: Gib z. B. „TestGarden1“ ein und bestätige mit ENTER. 4. Beobachte nach Bestätigung, ob ein neues Spielinterface geladen wird. 5. Prüfe im gardens Ordner, ob eine neue .map Datei erzeugt wurde. 6. Prüfe in der gardens_data.json, ob ein neuer Eintrag erzeugt wurde.	1. Das Spiel fragt erfolgreich die Vegetation ab (Fenster mit „Choose vegetation: …“). 2. Nach Eingabe des Namens wechselt das Spiel in den „Garten-Editor“-Bildschirm. 3. Im gardens Ordner liegt nun eine neue .map Datei. 4. In gardens_data.json existiert ein entsprechender Eintrag.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Testfall 16: Garten laden (Load Garden)

Testfall-ID: ITVG03

Testziel: Prüfung der Funktionalität zum Laden eines existierenden Gartens:

- Auswahl im Hauptmenü („Load Garden“).
- Dateiauswahl-Dialog via load_garden_dialog().
- Korrekte Wiederherstellung der Vegetation und GardenObjects über create_garden_objects().

Voraussetzungen:

- Spiel befindet sich im Hauptmenü.
- Mindestens eine .map-Datei liegt in MAP_FOLDER_PATH vor (z. B. „MyDesert.map“).
- gardens_data.json enthält einen entsprechenden Eintrag

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Klicke im Hauptmenü auf „Load Garden“. 2. Wähle in der angezeigten Liste eine .map aus. 3. Beobachte, ob der Gartenbildschirm geöffnet wird. 4. Stelle sicher, dass die geladenen Objekte (z. B. Kakteen, Büsche etc.) zum Vegetationstyp passen (z.B. „Desert“).	1. Nach der Auswahl von „MyDesert.map“ zeigt das Spiel den entsprechenden Garten an. 2. Die Vegetation stimmt mit „Desert“ überein (Sandhintergrund, Objekte der Wüste im Inventar). 3. Keine Fehlermeldungen in der Konsole.	1. erfüllt 2. erfüllt 3. erfüllt

Testfall 17: Platzieren eines Objekts mit ausreichenden und nicht ausreichenden Punkten

Testfall-ID: ITVG04

Testziel: Verifizieren, dass das Platzieren eines Objekts funktioniert, wenn genügend „available_points“ vorhanden sind. Integration zwischen Garden-Objekt, Punktelogik aus JSON-Datei und Inventar-Darstellung.

Voraussetzungen:

- Ein geladener oder neu erstellter Garten ist geöffnet.
- available_points ist im JSON_FILE auf einen Wert ≥ 10 gesetzt (zum Testen ausreichend hoch).
- Der Nutzer ist im Garten-Editor, ein Objekt (z. B. tree mit Kosten=8) steht im Inventar zur Verfügung.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Stelle sicher, dass available_points ≥ 2 2. Klicke in der Inventarleiste auf ein Objekt das 2 Punkte kostet. 3. Klicke anschließend irgendwo auf die freie Gartenfläche. 4. Beobachte, ob das Objekt platziert wird. 5. Prüfe die neue Punktzahl (z. B. „Points: 8“, wenn man 10 hatte und 2 ausgibt). 6. Klicke in der Inventarleiste auf ein Objekt das mehr Punkte kostet, als im Punktekonto verfügbar sind. 7. Klicke anschließend irgendwo auf die freie Gartenfläche. 8. Beobachte, ob das Objekt platziert wird, ob eine Fehlermeldung erscheint und prüfe die Punktzahl	-- ausreichende Punkte -- 1. Das ausgewählte Objekt erscheint auf der angeklickten Position des Gartens. 2. Punktestand sinkt korrekt um die Objektkosten (z. B. von 10 auf 8). 3. Keine Fehlermeldung 4. Die .map-Datei wird gespeichert (logisch oder im Dateisystem überprüfbar). -- nicht ausreichenden Punkte -- 5. Das ausgewählte Objekt erscheint nicht ! auf der angeklickten Position des Gartens. 6. Punktestand bleibt bestehen. 7. Fehlermeldung „Not enough points“ wird in der Konsole angezeigt	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt 5. erfüllt 6. erfüllt 7. erfüllt

Systemtests

Testfall 1: Gesamte Funktionsabfolge als Endanwender

Testfall-ID: ST01

Testziel: Prüfen, ob die Kernabläufe (Projekte anlegen, Zeit tracken, Punkte erhalten, Daten speichern/laden, Anwendungen wechseln, Garten laden und Objekte platzieren) in einer durchgängigen Endnutzer-Situation fehlerfrei funktionieren.

Voraussetzungen:

- Anwendung ist installiert/ausführbar.
- Eine leere oder Standard-Datenbank ist vorhanden.
- Vorhandensein von Standard-Dateien (Images, JSON).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Start der Anwendung • Stelle sicher, dass das Hauptfenster erscheint. 2. Projekt anlegen • Lege ein neues Projekt an (z. B. „Mein Systemtest-Projekt“). • Fülle Name, Beschreibung, Kategorie und Start/Enddatum. 3. Stopwatch starten • Stelle den Modus auf „Stopwatch“. • Starte, pausiere, setze fort, und stoppe schließlich die Uhr. 4. Punktesystem prüfen • Beobachte, ob nach einigen gesammelten Minuten die Punkte hochgezählt werden. 5. Kompletten Pomodoro-Zyklus durchlaufen • Schalte den Modus auf Pomodoro. • Eingabe gültiger Zeiten (z. B. Work 00:25:00, Break 00:05:00). • Starte, warte den Work-Abschnitt ab, prüfe automatischen Wechsel zur Break-Phase. 6. Projekt-Zeit prüfen • Wechsle zum Projekt, sieh dir den Zeitfortschritt im Kreis (z. B. 15 min) an.	1. Anwendung bleibt stabil, ohne Fehlermeldungen / Abstürze. 2. Alle Eingaben werden korrekt übernommen und angezeigt. 3. Punkte, Projektzeit und Projektinfos, sowie Gartendaten sind konsistent. 4. Keine spürbaren Verzögerungen bei normalen Interaktionen.	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

• Füge manuell weitere Minuten hinzu und verifiziere Diagramm-Update. 7. Daten persistieren • Schließe die Anwendung. • Starte sie erneut. Prüfe, ob die Projekte, Punktestände, Timer-Einstellungen und Texte in der Oberfläche geladen sind. 8. Über Button „TO THE GARDENS" in die Spielkomponente wechseln 9. Neuen Garten anlegen • Prüfen ob Punktestand korrekt • Objekte platzieren 12. mit Escape-Taste ins Menü navigieren 13. Zurück zur Produktivanwendung wechseln und wieder zu den Gärten zurückwechseln. 14. Erstellten Garten laden • Prüfen ob Objekte platziert • Prüfen ob Punktestand korrekt		

Testfall 2: Performance- und Ressourcen-Test (manuell)

Testfall-ID: ST02

Testziel: Untersuchen der Reaktionsgeschwindigkeit und Hardwareauslastung (CPU, RAM) unter typischer und erhöhter Last.

Voraussetzungen:

- Anwendung ist installiert/ausführbar.
- Windows Task-Manager (oder ein ähnliches Monitoring-Tool) ist offen, um CPU-/RAM-Auslastung zu beobachten.

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Normalbetrieb • Starte die Anwendung; beobachte RAM-Verbrauch im Task-Manager. • Führe typische Aktionen aus (Stopwatch, Timer starten/stoppen, Projekt anlegen/löschen) und notiere grob CPU-/RAM-Peaks. 2. Stressphase • Wechsle schnell zwischen Pomodoro, Timer und Stoppuhr. • Erstelle mehrere Projekte in schneller Abfolge. • Öffne und schließe ggf. das Fenster (sofern Fenster-/Minimieren-Funktionen existieren). 3. Beobachtung • Achte auf Ruckler oder Verzögerungen in der GUI (z. B. Eingabeverzögerungen). • Prüfe, ob CPU-Last dauerhaft unverhältnismäßig hoch wird. 4. Über Button „TO THE GARDENS" in die Spielkomponente wechseln 5. Garten laden 6. Objekte platzieren 7. Schnell mehrmals Gärten schließen und wieder laden	1. Das Programm bleibt reaktionsschnell, kein Einfrieren oder starker Ressourcenanstieg. -- für beide Anwendungen jeweils: -- 2. CPU-Auslastung steigt eventuell kurzzeitig bei Diagramm-Updates, normalisiert sich aber zügig. 3. RAM-Verbrauch bleibt im erwarteten Rahmen (< einige hundert MB, je nach Projekt).	1. erfüllt -- Produktivanwendung -- 2. keine merkbare Mehrbelastung der CPU 3. etwa 50-80 MB RAM-Verbrauch -- Spielkomponente – 2. beim Laden eines Gartens leicht spürbare Mehrbelastung der CPU 3. etwa 80-100MB RAM-Verbrauch

Testfall 3: Hintergrundbetrieb und Minimieren

Testfall-ID: ST03

Testziel: Validieren, dass die Funktionen der Produktivanwendung wie der Timer oder die Stoppuhr weiterlaufen, wenn das Fenster minimiert wird oder in den Hintergrund wechselt.

Voraussetzungen:

- Produktivanwendung läuft normal im Vordergrund

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Start die Zeitmessung in einem beliebigen Modus. 2. Starte parallel eine Stoppuhr auf dem Smartphone oder Ähnlichem. 2. Minimiere die Anwendung oder wechsle zu einer anderen Anwendung. 3. Warte ca. 11–12 Minuten. 4. Maximiere die Anwendung wieder. 5. Über Button „TO THE GARDENS" in die Spielkomponente wechseln 6. Einen beliebigen Garten laden oder erstellen 7. Neues Objekte platzieren 8. Minimiere die Anwendung 9. Warte 1-2 Minuten 10. Maximiere die Anwendung	1. Die im Hintergrund weitergelaufene Zeit erscheint im Zeit-Label (z. B. 00:01:30). 2. Keine Fehlermeldung oder Stopp der Zeitmessung durch Minimieren. 3. Punkte-Berechnung (alle 10 Minuten) funktioniert weiterhin, wenn es im Hintergrund erreicht wird. 4. Garten bleibt geöffnet und platziertes Objekt bleibt bestehen	1. erfüllt 2. erfüllt 3. erfüllt 4. erfüllt

Statische Tests

Während der Entwicklung der verschiedenen Python Skripte und Komponenten wurden die Code-Analyse-Tools Pylance und Flake8 verwendet. Pylance wurde hauptsächlich eingesetzt als Echtzeitunterstützung und Flake8 zur Überprüfung der Codequalität und Einhaltung von PEP 8.

Blackbox-Tests

Aus Zeitgründen wurden keine Blackbox-Tests durchgeführt. Dies ist eine wichtige Methode, um Fehler in der Funktionalität und im Verhalten der Software zu identifizieren und einen besseren Einblick in die Gestaltung der Benutzeroberfläche und der Interaktionen zu bekommen. Daher wäre dies eine gute Möglichkeit, um weitere Verbesserungsmöglichkeiten der Anwendung zu identifizieren. Als Hilfe kann folgender Testfall dienen:

Testfall: Usability-Test durch Benutzer

Testziel: Überprüfen, wie „externe“ Testpersonen den Workflow wahrnehmen.

Voraussetzungen:

- Ein Test-User (nicht Entwickler) ist verfügbar.
- Kurze Einführung in die Hauptfunktionen (Projekte, Timer, Punkte).

Eingabedaten/Aktionen	*Erwartetes Ergebnis*	*Tatsächliches Ergebnis*
1. Der Test-User bedient die Anwendung auf eigene Faust. 2. Der Test-User dokumentiert: Verständnisfragen („Wo erstelle ich ein neues Projekt?“). Subjektive Wahrnehmungen wie bspw. Reaktionszeiten und Optik der Oberfläche. 3. Gefundene Bugs oder unklare Fehlermeldungen.	1. Nutzer findet sich intuitiv zurecht, kann bspw. Zeit erfassen, Projekte anlegen und zu den virtuellen Gärten wechseln. 2. Es gibt keine großen Hindernisse (z. B. unklare Eingabefelder oder Buttons). 3. Feedback zur GUI-Leistung und -Optik ist überwiegend positiv.	…