فهرست مطالب

مقدمه و نکات مهم

هدف نهایی این پروژه، طراحی و پیاده‌سازی یک **محیط برنامه‌نویسی بصری (Visual Programming Environment)** مشابه نرم‌افزار معروف **Scratch** است.[1] به بیان دقیق‌تر، در این پروژه شما **کاربر** اسکرچ نیستید، بلکه **سازنده** نرم‌افزاری شبیه به آن هستید. شما باید با استفاده از زبان C++ و کتابخانه SDL2 برنامه‌ای توسعه دهید که قابلیت‌هایی نظیر انتخاب، جابجایی (Drag & Drop) و اتصال بلوک‌های دستوری و در نهایت اجرای آن‌ها را فراهم کند.
شما می‌توانید جهت الهام‌گیری و بررسی عملکرد دقیق محیط، به وب‌سایت MIT Scratch مراجعه نمایید.

نکته فنی: پیاده‌سازی این پروژه با استفاده از کتابخانه SDL2 انجام خواهد شد. اما استفاده از ورژن جدیدتر کتابخانه مذکور (یعنی SDL3) بلامانع است.(توجه شود که این کار موجب برتری خاصی در نمره‌دهی نخواهد شد.)

بارم‌بندی

- **نمره اصلی:** ۳ نمره
- **نمره جبرانی:** ۱ و نیم نمره
- **نمره امتیازی برای معدود گروه‌های برتر (Bonus):** نیم نمره

جزئیات بیشتر بارم بندی در اینجا موجود است. آیدی تلگرام همیاری که هر بخش را نوشته است هم قرار داده شده است تا در صورت ابهام از ایشان بپرسید.

گزارش برنامه‌ریزی انجام پروژه (Checkpoint)

برای جلوگیری از موکول کردن پروژه به دقیقه نود توسط شما، لازم است نحوه تقسیم بندی وظائف بین اعضای گروه و در حالت کلی برنامه ریزی که برای انجام پروژه انجام دادید را در قالب یک گزارش کوتاه تا روز ۲۸ دی ۱۴۰۴ تحویل بدهید.
این بخش به تنهایی نمره‌ای ندارد، اما عدم انجام آن منجر به محرومیت از کسب نمره جبرانی و یا امتیازی خواهد شد.

موعد تحویل نهایی

روز ۲۲ بهمن سال ۱۴۰۴، توجه شود که این موعد به هیچ عنوان امکان تمدید شدن و به تعویق افتادن ندارد.

۱.۰ اهمیت و نحوه استفاده از گیت‌هاب (GitHub)

با توجه به ویدئوهای آموزشی، کارگاه و جلسات رفع اشکال برگزار شده توسط گروه محترم حل تمرین، استفاده **مؤثر و مستمر** از گیت‌هاب در این پروژه یک گزینه اختیاری نیست، بلکه **اجباری** است. استفاده مؤثر به معنای درگیر کردن گیت در تمام مراحل توسعه پروژه است؛ یعنی شما باید به طور منظم Commit، Push و Branchهای منطقی داشته باشید. حتی برای پروژه‌های تک‌نفر یا دونفره، رعایت اصول ورژن کنترل (Version Control) الزامی است. نباید صرفاً در انتهای کار، کل پروژه را یک‌جا آپلود کنید تا فقط به ظاهر نشان دهید که از گیت استفاده کرده‌اید.

موارد زیر به عنوان «دور زدن قوانین گیت» و عدم فعالیت موثر شناسایی می‌شوند:

[1] اسکرچ پلتفرمی است که در آن برنامه‌نویسی نه با تایپ متن، بلکه با اتصال بلوک‌های گرافیکی انجام می‌شود.

- ❋ **آپلود یک‌جا (Bulk Upload):** بارگذاری تمام فایل‌های نهایی در یک یا دو کامیت در روزهای آخر.
- ❋ **کامیت‌های بی‌معنی:** استفاده از پیام‌های کامیت مبهم و تکراری مانند "update"، "fix" یا "." بدون توضیح تغییرات.
- ❋ **فعالیت تک‌بعدی:** در گروه‌های چندنفره، اگر فقط یک نفر تمام کامیت‌ها را انجام دهد و اکانت نفر بقیه غیرفعال باشد.

تأثیر در نمره‌دهی

- ❋ **عدم استفاده از گیت‌هاب:** نمره پروژه **صفر** لحاظ خواهد شد.
- ❋ **عدم استفاده مؤثر (رعایت نکردن اصول فوق):** حداکثر ۱ نمره از ۳ نمره اجباری قابل دریافت است و نمره جبرانی و یا متیازی به هیچ وجه تعلق نخواهد گرفت.

۲.۰ اهمیت کدنویسی تمیز (Clean Code)

رعایت اصول کدنویسی تمیز نقش حیاتی در کاهش هزینه‌های زمانی و انسانی تیم‌های فنی دارد. تحقیقات نشان می‌دهد که توسعه‌دهندگان معمولاً ۳۰ تا ۴۰ درصد زمان خود را صرف نوشتن کدهای جدید می‌کنند، در حالی که حدود ۶۰ درصد زمان آن‌ها صرف خواندن، نگهداری و اصلاح (Debug) کدهای موجود می‌شود. کد شما باید **شیوا و خوانا** باشد؛ به گونه‌ای که هم‌گروهی یا ارزیاب پروژه، برای درک عملکرد آن متحمل زجر و اتلاف وقت نشود.

چند نکته ساده اما کلیدی:

- ❋ **نام‌گذاری استاندارد:** نام کلاس‌ها، توابع و متغیرها باید دقیقاً بازتاب‌دهنده عملکرد آن‌ها باشد (مانند `execBlockSequence` به جای `run`). این کار جستجو و درک منطق کد را آسان می‌کند.
- ❋ **منطق و طراحی:** پیش از کدنویسی، تفکر الگوریتمی و طراحی مسیر پیاده‌سازی بسیار مهم است. مطالعه و مشاهده کدهای استاندارد می‌تواند در شکل‌گیری این ذهنیت به شما کمک کند.

تأثیر در نمره‌دهی

هدف ما سخت‌گیری بی‌مورد نیست؛ اما اگر کد به قدری نامرتب و "کثیف" باشد که خوانایی آن برای ارزیاب غیرممکن شود، **ضریب ۵۰ درصدی** بر نمره نهایی اعمال خواهد شد. بنابراین با رعایت نکات ساده‌ای مثل نام‌گذاری بامعنا و کامنت‌گذاری مناسب، به خود و تیم ارزیابی‌کننده کمک کنید.

۳.۰ اهمیت کار گروهی

در گروه‌های چندنفره، مشارکت فعال تمامی اعضا **اجباری** است. پروژه باید حاصل تلاش مشترک باشد.

تأثیر در نمره‌دهی

در صورتی که اعضای گروه نسبت به کم‌کاری، انفعال یا عدم همکاری یکی از اعضا شکایت داشته باشند، موضوع توسط تیم ارزیابی‌کننده بررسی می‌شود. در صورت اثبات ادعا، فرد مذکور **هیچ نمره‌ای** از بخش پروژه دریافت نخواهد کرد.

۴.۰ خط‌مشی استفاده از هوش مصنوعی (AI Policy)

شما مجاز هستید برای یادگیری، رفع اشکال و دریافت راهنمایی از دوستان، منابع آنلاین و ابزارهای هوش مصنوعی (مانند ChatGPT، Copilot و ...) استفاده کنید. اما توجه داشته باشید که این ابزارها باید نقش **مشاور** را داشته باشند، نه **کدنویس**! کد نهایی باید توسط خود شما نوشته و درک شده باشد. اگر کدی مستقیماً توسط هوش مصنوعی تولید و کپی شود، مشابه تقلب در تمرین و کوییز با آن برخورد شده و اثر منفی بر نمره خواهد داشت. از آنجا که کار گروهی است، لطفاً با هم‌گروهی‌های خود صادق باشید؛ زیرا جریمه استفاده نادرست از AI گریبان‌گیر **کل گروه** خواهد شد (تر و خشک با هم می‌سوزند).

مثال‌هایی برای درک مرز استفاده صحیح و غلط:

استفاده صحیح (AI به عنوان مشاور): "من می‌خواهم قابلیت *Drag & Drop* برای بلوک‌ها را پیاده‌سازی کنم. چگونه می‌توانم در *SDL2* تشخیص دهم که موس روی یک مستطیل کلیک کرده و نگه‌داشته شده است؟ (بدون نوشتن کد کامل، منطق را توضیح بده)"
(در این حالت شما به دنبال یادگیری منطق و رفع ابهام هستید.)

استفاده نادرست (AI به عنوان کدنویس): "یک کلاس کامل به زبان *C++* بنویس که یک بلوک دستوری گرافیکی بسازد، به بقیه بلوک‌ها بچسبد و وقتی دکمه اجرا زده شد، دستور را اجرا کند."
(در این حالت شما فرآیند تفکر و کدنویسی را به طور کامل حذف کرده‌اید.)

توضیحات هر بخش

۱ منوی فایل

۱.۱ ذخیره‌سازی و Load کردن فایل

بعد از پیاده‌سازی بخش‌های مورد نیاز برنامه، طبیعتاً به قابلیت ذخیره‌سازی اطلاعات نیاز خواهید داشت. برای این کار ابتدا منویی تعریف کنید که قابلیت‌های Save کردن، Load کردن و ساخت پروژه جدید در آن در دسترس باشد.

- **New Project:** پاکسازی محیط کار و آماده‌سازی برای شروع یک پروژه جدید.
- **Save Project:** ذخیره وضعیت فعلی بلاک‌ها و اشیاء روی حافظه.
- **Load Project:** بازخوانی اطلاعات از فایل و بازسازی محیط برنامه.

۲.۱ راهنمای پیاده‌سازی سیستم ذخیره‌سازی (Save/Load) پروژه

در حال حاضر وقتی برنامه‌ی خود را می‌بندید، تمام بلاک‌هایی که چیده‌اید از بین می‌روند چون در حافظه موقت (RAM) هستند. برای ماندگار کردن پروژه‌ها، باید آن‌ها را روی حافظه دائمی (هارد دیسک) ذخیره کنیم.

۳.۱ مفاهیم پایه: سریالایز و دی‌سریالایز چیست؟

قبل از شروع کدنویسی، باید با دو مفهوم کلیدی در دنیای نرم‌افزار آشنا شوید:

- **سریالایز کردن (Serialization):** به فرآیند تبدیل اشیاء زنده در حافظه برنامه (مثل بلاک‌ها، مختصات و مقادیر) به یک فرمت متنی یا باینریِ قابل ذخیره روی دیسک گفته می‌شود. مثل این است که نقشه‌ی دقیق ساخت یک قلعه لگویی را روی کاغذ بنویسید.
- **دی‌سریالایز کردن (Deserialization):** فرآیند معکوس است؛ یعنی خواندن آن متن از روی فایل و تبدیل دوباره‌ی آن به اشیاء زنده در محیط برنامه. مثل این است که از روی آن نقشه، دوباره قلعه لگویی را بسازید.

۴.۱ ابزارهای مورد نیاز در C++

در زبان C++ ابزارهای مورد نیاز برای پیاده‌سازی توابع ذخیره کردن و Load کردن فایل‌ها در اختیار شما قرار داده شده است (که در درس با آن‌ها آشنا شده‌اید):

- #include <fstream> برای کار با فایل‌ها و خواندن/نوشتن روی آن‌ها
- #include <string> برای کار با رشته‌ها (String) و رمزگشایی/پردازش آن‌ها

برای ساختار فایل‌های ذخیره‌شده نیز می‌توانید از یک فرمت دلخواه که متناسب با پیاده‌سازی پروژهٔ خودتان تنظیم می‌کنید استفاده کنید. مثلاً برای ذخیره‌سازی یک Sprite می‌توانید از قالب زیر استفاده کنید:

Name|xPosition|yPosition

که در واقع همان سریالایز کردن شخصیت برنامه متناسب با مشخصه‌های تعریف‌کنندهٔ آن است.

توجه: این صرفاً یک پیشنهاد خام است و شما برای پروژهٔ خود بهتر است ساختاری متناسب با اشیاء و نیازهای داخل پروژهٔ خود پیاده‌سازی کنید.

۲ منوی راهنما

۱.۲ راهنمای Debug و معرفی سیستم

در توسعه‌ی یک مفسر (Interpreter)، بزرگترین چالش این است که بدانیم در «مغز» سیستم چه می‌گذرد. از آنجا که ما کدِ C++ را اجرا نمی‌کنیم، بلکه داریم داده‌هایی را پردازش می‌کنیم که حکم «دستور» را دارند، ابزارهای دیباگ معمولی (مثل دیباگر Visual Studio) کمک زیادی به فهم منطقِ برنامه‌ی کاربر نمی‌کنند. بنابراین ما باید ابزارهای نظارتی خودمان را بسازیم.

۲.۲ سامانه ثبت وقایع (System Logger) - جعبه سیاه

این سیستم دقیقاً باید مثل «جعبه سیاه هواپیما» عمل کند. اگر برنامه ناگهان متوقف شد یا کاراکتر غیب شد، باید بتوانیم با چک کردن لاگ‌ها، آخرین لحظات حیات سیستم را بازسازی کنیم.

- ❇ **چیزی فراتر از cout معمولی:** نباید در هر جای کد به صورت پراکنده cout بنویسیم. بهتر است یک تابع مرکزی (مثلاً LogEvent) داشته باشیم که تمام بخش‌های سیستم گزارش‌هایشان را به آن تحویل دهند.
- ❇ **ساختار استاندارد گزارش:** هر خط لاگ بهتر است شامل ۵ بخش کلیدی باشد تا قابل ردیابی باشد:
  - **زمان/نوبت اجرا:** (مثلاً: در سیکل شماره ۵۴۳)
  - **موقعیت:** (مثلاً: خط ۵ برنامه کاربر)
  - **عامل:** (مثلاً: دستور MOVE یا دستور IF)
  - **عملیات:** (مثلاً: تغییر X)
  - **داده:** (مقدار قبلی چه بود → مقدار جدید چیست)
- ❇ **نمونه خروجی:**

[Cycle:120] [Line: 04] [CMD: MOVE] -> Changed X from 100 to 110

- ❇ **سطح‌بندی گزارش‌ها (Log Levels):** سیستم باید بتواند اهمیت پیام‌ها را تفکیک کند:
  - INFO: اتفاقات عادی (مثل حرکت کردن).
  - WARNING: اتفاقات مشکوک اما غیرمخرب (مثل تلاش برای حرکت وقتی به دیوار چسبیده است).
  - ERROR: اتفاقات خطرناک (مثل تلاش برای دسترسی به متغیری که وجود ندارد).

۳.۲ حالت اجرای گام‌به‌گام (Step-by-Step Mode)

این قابلیت به کاربر (و توسعه‌دهنده) اجازه می‌دهد زمان را متوقف کند. این فقط یک «مکث» ساده نیست، بلکه تغییرِ ریتمِ پردازشگر است.

- ❇ **منطق «توقف در لحظه»:** در حلقه اصلی برنامه (Main Loop)، قبل از اینکه دستور بعدی پردازش شود، سیستم باید شرط IsDebugMode را چک کند.

- **چرخه تعامل (Interaction Loop):** اگر حالت دیباگ فعال بود، سیستم وارد یک حلقه داخلی کوچک می‌شود و منتظر می‌ماند تا کلیدی زده شود (مثلاً Space). در این مدت، رندر کردن گرافیک نباید متوقف شود (تا کاربر صفحه سفید نبیند)، اما منطق برنامه (Logic) فریز می‌شود.
- **اجرای دقیقِ یک دستور:** به محض فشردن کلید، سیستم دقیقاً یک دستور را اجرا می‌کند، متغیرها را آپدیت می‌کند و دوباره به حالت انتظار می‌رود.
- **شفافیت وضعیت (State Visibility):** در این حالت، نمایش مقادیر فعلی متغیرها (مثل X، Y، جهت، متغیرهای شمارنده‌ی حلقه) روی صفحه یا کنسول بسیار حیاتی است. کاربر باید ببیند که با زدن دکمه، کدام عدد تغییر کرد.

۴.۲ مدیریت خطا و ایمنی (Error Handling & Safety Nets)

سیستم ما باید در برابر اشتباهات کاربر «ضد ضربه» (Robust) باشد. اگر کاربر برنامه‌ای نوشت که منطقاً غلط است، سیستم ما نباید کرش کند (بسته شود). ما از استراتژی «شکستِ کنترل‌شده» (Graceful Failure) استفاده می‌کنیم.

۱.۴.۲ الف) محافظت از مرزها (Boundary Checks)

هر دستوری که تغییری در مختصات یا ظاهر ایجاد می‌کند، ابتدا باید توسط یک تابع «تأییدکننده» چک شود.

- **سناریو:** کاربر دستور می‌دهد «۱۰۰۰ قدم برو جلو».
- **واکنش سیستم:** محاسبه می‌کند که ۱۰۰۰ قدم باعث خروج از صفحه می‌شود. به جای اینکه کاراکتر را گم کند، یا حرکت را لغو می‌کند یا کاراکتر را دقیقاً روی لبه‌ی دیوار نگه می‌دارد (Clamping).

۲.۴.۲ ب) محافظت محاسباتی (Math Safeguards)

عملیات ریاضی خطرناک باید در یک تابع ایمن انجام شوند.

- **تقسیم بر صفر:** اگر کاربر نوشت $A = 10 / 0$، سیستم باید قبل از تقسیم، مخرج را چک کند. اگر صفر بود، عملیات را لغو کرده و یک هشدار واضح چاپ کند (مثلاً: «خطای ریاضی در خط ۱۰»).
- **جذر عدد منفی:** مشابه بالا، برای توابعی مثل sqrt باید ورودی چک شود.

۳.۴.۲ ج) محافظت در برابر حلقه‌های مرگ (Infinite Loop Watchdog)

یکی از خطرناک‌ترین باگ‌ها، حلقه‌های Forever یا While هستند که هرگز تمام نمی‌شوند و کل سیستم را قفل می‌کنند (Halt).

- **راهکار:** یک «شمارنده ایمنی» (Watchdog Counter) داشته باشید. اگر تعداد دستورات اجرا شده در یک فریم از حد مشخصی (مثلاً ۱۰۰۰ دستور) بیشتر شد ولی هنوز تصویر رفرش نشده بود، سیستم باید تشخیص دهد که در یک حلقه بی‌نهایت گیر کرده است.
- **رفتار پیشنهادی:** سیستم به زور از حلقه خارج شود یا برنامه را متوقف کند و پیام دهد: «تشخیص حلقه بی‌نهایت در خط □»

۳ Add Extension

۱.۳ دستورات ترسیمی (Pen) با رنگ سبز پررنگ

در این بخش از پروژه، هدف پیاده‌سازی زیرسیستم گرافیکی Pen است که به کاربر اجازه می‌دهد همزمان با حرکت کاراکترها، روی صفحه ترسیمات گرافیکی انجام دهد. این بخش شامل دو قسمت اصلی «مدیریت افزونه‌ها» و «منطق بلوک‌های کدنویسی» است.

۲.۳ رابط کاربری انتخاب افزونه (Extension Library)

برنامه باید مجهز به یک سیستم پویا برای اضافه کردن قابلیت‌های جانبی باشد. یعنی در گوشه‌ای از محیط کاربری یا نوار گزینه‌ها، باید دکمه‌ای تعبیه شده باشد که وظیفه فراخوانی کتابخانه افزونه‌ها را بر عهده داشته باشد. با کلیک بر روی این دکمه، باید محیط اصلی موقتاً غیرفعال شده و یک پنل تمام‌صفحه شامل لیست افزونه‌ها نمایش داده شود. در این لیست، افزونه Pen باید به عنوان یک گزینه قابل انتخاب وجود داشته باشد. با کلیک بر روی افزونه Pen، این ماژول باید فعال شده و یک دسته‌بندی جدید با رنگ سبز پررنگ به انتهای لیست دستورات (در بخش Code) اضافه شود.

۳.۳ عملکرد بلوک‌های کنترلی قلم

پس از فعال شدن افزونه، مجموعه‌ای از دستورات در لیست کدها ظاهر می‌شوند که رفتار قلم را کنترل می‌کنند:

- **Erase All**: این بلوک وظیفه دارد تمامی اثرات قبلی قلم، خطوط رسم‌شده و تصاویر ثبت‌شده توسط Stamp را به صورت آنی از کل صفحه پاک کند.
- **Stamp**: این بلوک تصویری از وضعیت فعلی کاراکتر (Sprite) را در همان مختصات لحظه‌ای، روی خروجی گرافیکی چاپ می‌کند؛ به طوری که با حرکت کردن کاراکتر، آن تصویر در جای خود ثابت می‌ماند.
- **Pen Down و Pen Up**: مدیریت ترسیم بر عهده این دو بلوک است. زمانی که دستور Pen Down اجرا شود، قلم در وضعیت فعال قرار می‌گیرد و با جابه‌جایی کاراکتر در صفحه، باید مسیری ممتد پشت سر آن رسم شود. در مقابل، دستور Pen Up رسم خط را متوقف می‌کند تا کاراکتر بتواند بدون ایجاد اثر گرافیکی جابه‌جا شود.

۴.۳ مدیریت رنگ و ویژگی‌های بصری (Color & Attributes)

بخش مهمی از این افزونه، قابلیت تنظیم دقیق رنگ قلم است. در این بخش دو نوع بلوک Set و Change وجود دارد:

- **منوی انتخاب ویژگی (Dropdown)**: در بلوک‌های تنظیمی، باید یک منوی کشویی تعبیه شده باشد که کاربر بتواند مشخص کند قصد تغییر کدام ویژگی را دارد. این منو شامل گزینه‌هایی مثل Color (رنگ اصلی)، Saturation (اشباع رنگ) و Brightness (روشنایی) است.
- **بلوک‌های Set**: مقدار ویژگی انتخابی (مثلاً Brightness) را دقیقاً برابر عددی که در باکس عددی سفید رنگ مقابل آن نوشته شده تنظیم می‌کنند.
- **بلوک‌های Change**: مقدار فعلی ویژگی را به میزان عددی که در باکس ورودی نوشته شده افزایش یا کاهش می‌دهند (یعنی: مقدار فعلی + مقدار ورودی).

- **انتخاب مستقیم رنگ:** یک بلوک اختصاصی برای تعیین مستقیم رنگ وجود دارد که دارای یک دایره رنگی است و به کاربر اجازه می‌دهد رنگ قلم را به صورت بصری انتخاب کند.

۵.۳ تنظیم ضخامت و ابعاد قلم (Pen Size)

برای کنترل ضخامت خطوط ترسیمی، دو بلوک اختصاصی در نظر گرفته شده است:

- **Set Pen Size to:** مقدار ضخامت قلم را دقیقاً برابر با عدد درج‌شده در باکس ورودی قرار می‌دهد.
- **Change Pen Size by:** ضخامت فعلی را به اندازه عدد ورودی بزرگتر یا کوچکتر می‌کند.

تمامی این تغییرات باید به صورت آنی و در لحظه در خروجی گرافیکی اعمال شوند.

۶.۳ الزامات نهایی و رفتار سیستم

- **ماندگاری خروجی:** خطوط رسم‌شده نباید با هر فریم از بین بروند و تنها با دستور Erase All پاک شوند.
- **اولویت نمایش:** خطوط قلم و تصاویر Stamp شده باید همیشه پشت سر کاراکترها و جلوی پس‌زمینه اصلی (Backdrop) قرار بگیرند.
- **اعمال آنی تغییرات:** تمام تغییرات (مثل تغییر رنگ یا سایز) باید بلافاصله پس از اجرای دستور، روی ترسیمات جدید اعمال شوند.

۴ منوی Code

۱.۴ دستورات حرکتی (آبی‌رنگ)

دستورات حرکتی وظیفه‌ی کنترل مکان، جهت و نحوه‌ی حرکت هر Sprite در صحنه‌ی اجرای برنامه را بر عهده دارند. در این پروژه هر Sprite حداقل دارای ویژگی‌های پایه‌ای زیر است:

- **موقعیت دوبعدی:** مختصات x و y
- **جهت حرکت:** direction
- **وضعیت نمایش در صحنه:** (نمایش داده شود/مخفی باشد)

۱.۱.۴ مواردی که باید در پروژه پیاده‌سازی شوند

بر این اساس، دستورات حرکتی زیر در پروژه پشتیبانی می‌شوند:

- **حرکت خطی** Sprite به اندازه‌ی مشخص (مثلاً move N steps)
- **چرخش** Sprite در جهت ساعت‌گرد و پادساعت‌گرد (مثلاً turn right/left)
- **تنظیم مستقیم موقعیت** Sprite روی صحنه (مثل go to x: y:)
- **تغییر نسبی مختصات** x و y (مثل change x/y by)
- **تنظیم جهت حرکت** به یک زاویه‌ی مشخص (مثل point in direction)
- **انتقال به موقعیت‌های از پیش تعریف‌شده** مثل:
  - مکان تصادفی (random position)
  - مکان نشانگر ماوس (mouse pointer)
- **تعامل با مرزهای صحنه:** رفتار ساده‌ای مانند توقف یا بازگشت Sprite از لبه‌های صحنه در نظر گرفته می‌شود.

۲.۱.۴ مواردی که لزومی به پیاده‌سازی ندارند

در این پروژه پیاده‌سازی قابلیت‌های پیشرفته‌ی مرتبط با حرکت که موجب افزایش قابل‌توجه پیچیدگی می‌شوند، ضروری نیست. از جمله:

- شبیه‌سازی دقیق فیزیک حرکت و برخورد
- برخورد پیکسلی یا محاسبات هندسی دقیق
- مدل‌سازی شتاب، اصطکاک یا حرکت غیرخطی
- پشتیبانی از فضاهای سه‌بعدی

۳.۱.۴ نکات نیازمند تعریف در مستندات

برخی مفاهیم در این بخش نیاز به تعریف صریح در مستندات پروژه دارند، از جمله:

- **واحد حرکت:** منظور از یک «قدم» یا step چیست؟
- **بازه‌ی مجاز مختصات صحنه:** حداقل/حداکثر x و y در محیط اجرای شما
- **تعریف زاویه‌ها:** زاویه‌ی صفر کدام جهت است و جهت مثبت چرخش چگونه تعریف می‌شود؟

۲.۴ دستورات نیلی‌رنگ (Looks)

در محیط برنامه‌نویسی Scratch، دستورات دستهٔ Looks مسئول کنترل ظاهر، وضعیت نمایشی و ویژگی‌های بصری Spriteها (اشیا) و Stage (صحنه) هستند. این دستورات شامل نمایش پیام‌ها، تغییر لباس (Costume) و پس‌زمینه (Backdrop)، تنظیم اندازه، اعمال افکت‌های گرافیکی و کنترل ترتیب قرارگیری Spriteها در صحنه می‌باشند.

۱.۲.۴ دستورات گفتن و فکر کردن (Say / Think)

- Say <text>
- Say <text> for <t> seconds
- Think <text>
- Think <text> for <t> seconds

این دستورات برای نمایش یک پیام متنی به‌عنوان «گفتن» یا «فکر کردن» Sprite استفاده می‌شوند.

- **نسخه بدون زمان:**
  - پیام باید به‌عنوان وضعیت جاری Sprite ذخیره شود.
  - پیام تا زمانی که با دستور دیگری تغییر نکند، باقی می‌ماند.
- **نسخه زمان‌دار:**
  - پیام فقط به مدت t ثانیه فعال است و پس از آن باید حذف شود.

نکته: تفاوت Say و Think در شکل گرافیکی نمایش آن‌ها است.

۲.۲.۴ دستورات لباس (Costume)

- Switch costume to <costume>
- Next costume

هر Sprite می‌تواند چندین لباس (Costume) داشته باشد.

- **Switch costume to:** لباس جاری Sprite باید به لباس مشخص‌شده تغییر کند.
- **Next costume:** Sprite به لباس بعدی خود (به‌صورت چرخشی) تغییر می‌کند.

نکته: در صورت نبودن لباس مورد نظر، یا پیام خطا چاپ شود یا دستور نادیده گرفته شود.

۳.۲.۴ دستورات پس‌زمینه (Backdrop)

- Switch backdrop to <backdrop>
- Next backdrop

این دستورات از لحاظ عملکردی مشابه دستورات Costume هستند، با این تفاوت که روی Stage و Backdropها اعمال می‌شوند.

۴.۲.۴ دستورات اندازه (Size)

- Change size by <value>
- Set size to <value> %

این دستورات اندازهٔ Sprite را تغییر می‌دهند. اندازه به صورت درصد مدل می‌شود.
نکته: مقدار پیش‌فرض را 100% در نظر بگیرید.

۵.۲.۴ افکت‌های گرافیکی (Graphic Effects)

- Change color effect by <value>
- Set color effect to <value>
- Clear graphic effects

این دستورات افکت‌های ظاهری Sprite را تغییر می‌دهند.
نکته: دستور Clear graphic effects تمام افکت‌ها را به مقدار پیش‌فرض (مثلاً صفر) برمی‌گرداند.

۶.۲.۴ نمایش و مخفی‌سازی (Show / Hide)

- Show
- Hide

این دستورات Sprite را قابل مشاهده یا مخفی می‌کنند.

۷.۲.۴ لایه‌ها (Layer Ordering)

- Go to <front / end> layer
- Go <forward / backward> <value> layers

این دستورات ترتیب قرارگیری Spriteها روی صحنه را تغییر می‌دهند.

۸.۲.۴ گزارشگرهای Looks (Reporter Blocks)

- Costume <number / name>
- Backdrop <number / name>
- Size

این دستورات مقدار فعلی هر ویژگی را برمی‌گردانند. همچنین با فعال کردن این دستورات، مقدار یا نام فعلی ویژگی موردنظر روی Stage نمایش داده می‌شود.

۳.۴ دستورات صوتی (Sound - بلوک‌های بنفش)

بخش صوتی مربوط به مدیریت و کنترل پخش صداها در سطح Sprite یا کل برنامه است.

۱.۳.۴ مواردی که باید در پروژه پیاده‌سازی شوند

در این پروژه هر Sprite می‌تواند شامل یک یا چند منبع صوتی باشد. دستورات اصلی این بخش شامل موارد زیر است:

- پخش یک صدای مشخص
- پخش صدا تا پایان اجرا
- توقف تمامی صداهای در حال پخش
- تنظیم یا تغییر سطح صدای پخش

۲.۳.۴ مواردی که لزومی به پیاده‌سازی ندارند

قابلیت‌های پیشرفته‌ی صوتی خارج از دامنه‌ی پروژه محسوب می‌شوند، از جمله:

- ضبط صدا از ورودی میکروفون
- اعمال افکت‌های صوتی پیچیده
- پردازش دیجیتال سیگنال (DSP) یا تغییر فرکانس صدا
- چندکاناله بودن سیستم صوتی

۳.۳.۴ نکات نیازمند تعریف در مستندات

- نحوه‌ی همزمانی اجرای صدا با سایر بخش‌های برنامه باید مشخص شود.
- بازه‌ی عددی تعیین‌شده برای حجم صدا (مثلاً حداقل و حداکثر مقدار مجاز) باید تعریف شود.

۴.۴ رویدادها (Events - بلوک‌های زرد)

بخش رویدادها نقش اصلی در پیاده‌سازی ساختار رویدادمحور برنامه را دارد و مسئول آغاز و هماهنگی اجرای اسکریپت‌ها است.

۱.۴.۴ مواردی که باید در پروژه پیاده‌سازی شوند

- شروع برنامه با یک رویداد کلی (مثلاً آغاز اجرای پروژه)
- واکنش به ورودی کاربر مانند فشردن کلید یا کلیک روی Sprite
- ارسال و دریافت پیام بین Spriteها از طریق مکانیزم broadcast

هر رویداد می‌تواند منجر به اجرای یک یا چند اسکریپت مرتبط شود و امکان ارتباط منطقی بین اجزای مختلف برنامه را فراهم کند.

۲.۴.۴ مواردی که لزومی به پیاده‌سازی ندارند

- اجرای همزمان واقعی مبتنی بر چندریسمانی (Multithreading)
- اولویت‌بندی پیشرفته‌ی رویدادها
- ارسال داده‌های پیچیده به همراه رویدادها
- زمان‌بندی دقیق بلادرنگ (Real-time Scheduling)

۳.۴.۴ نکات نیازمند تعریف در مستندات

- ترتیب اجرای اسکریپت‌های مرتبط با یک رویداد باید مشخص شود.
- هم‌زمان یا ترتیبی بودن اجرای رویدادها باید شفاف تعریف شود.
- نحوه‌ی مدیریت چند گیرنده برای یک پیام broadcast باید مشخص شود.

۵.۴ منطق دستورات کنترلی (Control Flow Logic)

این بخش پیچیده‌ترین قسمت ماجراست چون «ترتیب خطی» اجرا را تغییر می‌دهد.

۱.۵.۴ پیش‌نیاز مهم: اسکن اولیه (Pre-processing)

قبل از اینکه برنامه شروع به اجرا کند (وقتی دکمه Run زده شد)، سیستم شما باید یک بار کل لیست دستورات را سریع بخواند. هدف این مرحله پیدا کردن جفت‌های «شروع/پایان» است. مثلاً اگر در خط ۵ دستور If داریم، سیستم باید بگردد و End If مربوط به آن را پیدا کند (مثلاً خط ۸) و این آدرس را به عنوان مقصد پرش در اطلاعات همان خط ذخیره کند.

۲.۵.۴ منطق دستورات خاص

۱) دستور «صبر کن» (Wait)

- **منطق:** این دستور از توابع زمانی سیستم عامل استفاده می‌کند تا اجرای خط بعدی را برای چند میلی‌ثانیه/ثانیه به تأخیر بیندازد. در این مدت کل برنامه «فریز» می‌شود.

۲) دستور «تکرار کن» (Repeat)

❊ **چالش:** چطور بفهمیم چند بار تکرار کرده‌ایم؟ مخصوصاً اگر یک حلقه داخل یک حلقه دیگر باشد؟

❊ **راهکار:** استفاده از مفهوم «پشته» (Stack).

- وقتی به «شروع تکرار» می‌رسیم، تعداد تکرار را در پشته ذخیره می‌کنیم.
- وقتی به «پایان تکرار» می‌رسیم، مقدار بالای پشته را یکی کم می‌کنیم.
- اگر مقدار هنوز بزرگتر از صفر بود، شمارنده خط (PC) به خط شروع برمی‌گردد.
- اگر صفر شد، مقدار از پشته حذف می‌شود و از حلقه خارج می‌شویم.

۳) دستور «برای همیشه» (Forever)

❊ **منطق:** ساده‌ترین نوع پرش است. وقتی مفسر به بلوک Forever رسید، بدون هیچ شرطی شمارنده خط (PC) را به خطِ شروع Forever برمی‌گرداند.

۴) دستور «اگر ... آنگاه» (If-Then)

❊ **منطق:** شرط بررسی می‌شود.

- اگر شرط درست بود: برنامه عادی ادامه می‌یابد (وارد بدنه If می‌شود).
- اگر شرط غلط بود: سیستم باید «پرش» کند به خطی که در مرحله اسکن اولیه پیدا شده (یعنی خط End If).

۵) دستور «صبر کن تا ...» (Wait Until)

❊ **منطق:** این دستور برنامه را در «همان خط» نگه می‌دارد.

- تا زمانی که شرط برقرار نشده (False است)، شمارنده خط (PC) زیاد نمی‌شود.
- به محض برقرار شدن شرط، اجازه عبور به خط بعد صادر می‌شود.

۶) دستور «توقف همه» (Stop All)

❊ **منطق:** این دستور متغیر isRunning (شرط حلقه اصلی سیستم) را غیرفعال (False) می‌کند. در نتیجه موتور اجرایی متوقف شده و برنامه به پایان می‌رسد.

۷) دستور «اگر ... آنگاه ... وگرنه» (If-Then-Else) این دستور کمی پیچیده‌تر از If ساده است چون دو مسیر مجزا دارد.

❊ **ساختار در حافظه:**

IF_START -> [True-Branch] -> ELSE -> [False-Branch] -> END_IF

❊ **منطق:**

- **حالت اول (شرط درست است):** سیستم وارد بخش درست می‌شود؛ اما وقتی به خط ELSE رسید، نباید آن را اجرا کند و باید «پرش» کند به END_IF.
- **حالت دوم (شرط غلط است):** سیستم همان ابتدا چون شرط غلط است باید «پرش» کند به خطِ بعد از ELSE (شروع بخش غلط).

۸) دستور «تکرار کن تا زمانی که ...» (RepeatUntil) این دستور شبیه Repeat است، با این تفاوت که به جای شمارش تعداد، با یک «شرط» کار می‌کند.

❊ **منطق:**

❊ قبل از ورود به حلقه، شرط چک می‌شود.

❊ اگر شرط برقرار بود (True): یعنی کار تمام شده است؛ سیستم باید به خطِ End Repeat پرش کند (خروج از حلقه).

❊ اگر شرط برقرار نبود (False): دستورات داخل حلقه اجرا می‌شوند. وقتی به پایان حلقه رسیدیم، سیستم دوباره PC را به خط شروع برمی‌گرداند تا شرط مجدداً بررسی شود.

۶.۴ دستورات حسگری (لاجوردی)

دستورات Sensing در Scratch برای دریافت اطلاعات از محیط اجرا، تعامل با کاربر، وضعیت Spriteها و زمان استفاده می‌شوند. این دستورات به برنامه امکان می‌دهند نسبت به ورودی‌ها، موقعیت‌ها، برخوردها و پاسخ‌های کاربر واکنش نشان دهد.

۱.۶.۴ تشخیص تماس (Touching)

❊ Touching <mouse pointer / edge / sprite>?

❊ Touching color <color>?

❊ Color <color1> is touching <color2>?

این دستورات برای تشخیص تماس یا برخورد Sprite استفاده می‌شوند.

۲.۶.۴ فاصله (Distance)

❊ Distance to <mouse pointer / sprite>

فاصله می‌تواند با فرمول اقلیدسی محاسبه شود و خروجی باید یک مقدار عددی باشد.

۳.۶.۴ پرسش و پاسخ (Ask / Answer)

❊ Ask <question> and wait

❊ Answer

دستور Ask یک سؤال متنی از کاربر می‌پرسد و اجرای برنامه تا دریافت پاسخ متوقف می‌شود. پاسخ وارد شده توسط کاربر باید ذخیره شده و با دستور Answer قابل دسترسی باشد.

۴.۶.۴ تشخیص ورودی‌های صفحه‌کلید و ماوس

- Key <key> pressed?
- Mouse down?
- Mouse x
- Mouse y

این دستورات وضعیت فعلی ورودی‌های کاربر را بررسی می‌کنند:

- Key <key> pressed? بررسی می‌کند آیا کلید مشخص‌شده در حال حاضر فشرده است یا نه.
- Mouse down? بررسی وضعیت کلیک ماوس
- Mouse x / Mouse y موقعیت فعلی ماوس را برمی‌گرداند.

۵.۶.۴ کشیدن Sprite (Drag)

- Set drag mode <draggable / not draggable>

این دستور مشخص می‌کند که آیا Sprite می‌تواند با ماوس کشیده شود یا خیر.

۶.۶.۴ تایمر (Timer)

- Timer
- Reset timer

Timer زمان سپری‌شده از آخرین ریست را برمی‌گرداند و Reset timer مقدار تایمر را صفر می‌کند.

۷.۴ عملگرها (Operators)

با نیم‌نگاهی به عملگرهای موجود در C++ و همچنین عملگرهای پیاده‌سازی‌شده در محیط Scratch، برنامه شما باید تعدادی از عملگرهای موجود در Scratch را به‌صورت **اجباری** و باقی را به‌صورت **اختیاری/امتیازی** پشتیبانی کند.

۱.۷.۴ عملگرهای اجباری

- **عملگرهای ریاضی ساده:** جمع، تفریق، ضرب و تقسیم
- **عملگرهای مقایسه‌ای:** تساوی، کوچکتر/بزرگتر بودن
- **عملگرهای منطقی رایج:** AND، OR و NOT
- **عملگرهای استرینگی:** طول رشته، حرف n-ام یک رشته، و ادغام دو رشته

۲.۷.۴ عملگرهای امتیازی

- **عملگرهای تابعی ریاضی:** قدر مطلق، جذر، جزء صحیح و سقف، سینوس و کسینوس
- **باقیمانده و پیمانه‌گیری:** باقیماندهٔ یک مقسوم بر یک مقسومٌ‌علیه
- **عملگر منطقی دیگر:** XOR

۳.۷.۴ نکات پیاده‌سازی

- **عملگرهای ریاضی:** ورودی و خروجی عددی خواهند داشت و در غیر این صورت ورودی را نمی‌پذیرند.
- **عملگرهای مقایسه‌ای:** ورودی عددی و خروجی بولی (Boolean) خواهند داشت و در غیر این صورت ورودی را نمی‌پذیرند.
- **عملگرهای منطقی:** ورودی و خروجی بولی (Boolean) خواهند داشت و در غیر این صورت ورودی را نمی‌پذیرند.
- **عملگرهای استرینگی:** ورودی و خروجی رشته‌ای (String) خواهند داشت و در غیر این صورت ورودی را نمی‌پذیرند؛ **به‌جز** عملگر «پیدا کردن حرف n-امین رشته» که یک ورودی عددی نیز خواهد داشت.
- **ایندکس رشته‌ها:** برخلاف C++، در شمارش حروف رشته، ایندکس از **۱** شروع خواهد شد.
- **ورودی نامعتبر (امتیازی):** بررسی و مدیریت ورودی‌های نامعتبر امتیازی محسوب می‌شود (مانند: اندیس نامعتبر در رشته، عدد منفی زیر رادیکال، تقسیم بر صفر و).

۸.۴ متغیرها (Variables)

متغیرها برای نگه‌داری داده‌ها در برنامه استفاده می‌شوند. هر متغیر دارای یک نام بوده و مقدار آن در طول اجرای برنامه قابل تغییر است.
متغیرها از نظر مفهومی مشابه متغیرها در زبان C++ هستند؛ با این تفاوت که در این پروژه نیازی به تعیین نوع متغیر در زمان تعریف آن نیست و نوع متغیر به‌صورت خودکار از روی مقدار آن مشخص می‌شود (نوع پویا).

۱.۸.۴ ویژگی‌های متغیرها

- هر متغیر باید یک **نام یکتا** داشته باشد.
- متغیرها می‌توانند مقدار **عددی** یا **رشته‌ای** (String) نگه‌داری کنند.
- مقدار یک متغیر می‌تواند در طول اجرای برنامه تغییر کند.
- می‌توان از متغیرها در عملگرها، شرط‌ها و محاسبات استفاده کرد.

۲.۸.۴ نکات پیاده‌سازی

- متغیرها از نظر مفهومی مشابه متغیرهای C++ هستند، اما دارای **نوع پویا** (Dynamic Typing) می‌باشند.
- متغیرها ابتدا توسط برنامه (کاربر) **تعریف** می‌شوند و سپس قابل استفاده خواهند بود.

۳.۸.۴ موارد امتیازی

* پشتیبانی از نمایش مقدار متغیرها در زمان اجرا، در قالب یک جدول در بالای صفحهٔ اصلی اجرای برنامه.

۵ منوی تعیین ورودی-خروجی تابع

۱.۵ بلوک‌ها (توابع دلخواه) در Scratch و پیاده‌سازی آن‌ها در C++ با SDL

یکی از مهم‌ترین قابلیت‌های محیط برنامه‌نویسی Scratch، امکان تعریف توابع دلخواه (Custom Blocks) است. این قابلیت به کاربران اجازه می‌دهد مجموعه‌ای از بلاک‌ها را تحت یک نام مشخص گروه‌بندی کرده و آن‌ها را چندین بار در برنامه فراخوانی کنند. هدف این بخش، بررسی مفهوم تابع در Scratch و سپس ارائه‌ی یک مدل الگوریتمی برای پیاده‌سازی آن در C++ با استفاده از کتابخانه SDL است.

۱.۱.۵ تابع (Custom Block) در Scratch چیست؟

تابع دلخواه در Scratch یک بلاک قابل تعریف توسط کاربر است که شامل مجموعه‌ای از بلاک‌های دیگر می‌باشد. این مفهوم معادل **تابع** (Function) یا **رویه** (Procedure) در زبان‌های برنامه‌نویسی متنی است.

ویژگی‌های اصلی توابع دلخواه در Scratch:

- دارای نام مشخص
- قابلیت داشتن پارامتر (ورودی)
- اجرای ترتیبی بلاک‌های داخلی
- امکان فراخوانی چندباره از نقاط مختلف برنامه

توابع دلخواه به کاربران کمک می‌کنند:

- از تکرار بلاک‌ها جلوگیری کنند
- برنامه‌های بزرگ‌تر و ساخت‌یافته‌تر بسازند
- مفاهیم تفکر الگوریتمی را بهتر درک کنند

۲.۱.۵ نحوه ساخت تابع دلخواه در Scratch

برای ساخت یک تابع دلخواه در Scratch مراحل زیر انجام می‌شود:

۱. انتخاب دسته‌ی My Blocks
۲. کلیک روی گزینه‌ی Make a Block و وارد کردن نام تابع
۳. (اختیاری) اضافه کردن ورودی‌ها:
   - ورودی عددی (Number)
   - ورودی متنی (Text)
   - ورودی بولی (Boolean)
۴. تأیید ساخت بلوک

پس از این مراحل، Scratch به صورت خودکار دو بلاک ایجاد می‌کند:

- بلاک تعریف تابع

- بلاک فراخوانی تابع

توجه: در پروژه شما باید هر دو موردِ «بلاک تعریف تابع» و «بلاک فراخوانی تابع» در دسترس باشند، اما آزادی دارید آن‌ها را به هر شکل مناسب در پنجره برنامه قرار دهید؛ صرفاً باید دسترسی به آن‌ها ممکن باشد. برای مثال با کلیک بر بلوک فراخوانی بتوان یک نمونه از تابع را در جای دلخواه از کد قرار داد. همچنین باید بتوان با انتخاب بلوک تعریف تابع، تعریف اولیه را ویرایش کرد.

۳.۱.۵ ساختار بلوک تعریف تابع

بلوک تعریف تابع دارای ویژگی‌های زیر است:

- یک بلاک کلاه‌دار (Hat Block)
- دارای نام تابع
- شامل پارامترها
- دارای بدنه‌ای برای قرارگیری بلاک‌های دیگر

ساختار منطقی این بلاک مشابه ساختار زیر در زبان‌های متنی است:
define functionName(parameters) functionBody end
بدنه‌ی تابع شامل مجموعه‌ای از بلاک‌هاست که به صورت ترتیبی اجرا می‌شوند.

۴.۱.۵ بلاک‌هایی که می‌توانند داخل تابع قرار بگیرند

تقریباً تمام بلاک‌های Scratch می‌توانند داخل یک تابع دلخواه قرار گیرند، از جمله:

- **بلاک‌های حرکتی (Motion)**
- **بلاک‌های ظاهری (Looks)** مثل: say، show، hide، change costume
- **بلاک‌های کنترلی (Control)** مثل: repeat، forever، if، if else، wait
- **بلاک‌های محاسباتی و منطقی (Operators)** مثل: عملیات ریاضی، مقایسه‌ها، عملگرهای منطقی
- **بلاک‌های متغیرها (Variables)** مثل: set variable، change variable

نکته مهم: بلاک‌های رویدادی (Event Blocks) مانند:

- when green flag clicked
- when key pressed

نمی‌توانند داخل تابع قرار بگیرند، زیرا این بلاک‌ها نقطه‌ی شروع اجرای برنامه هستند و این مورد باید در کد شما رعایت شود.

۵.۱.۵ فراخوانی تابع در Scratch

پس از تعریف تابع، بلاک فراخوانی آن در بخش My Blocks ظاهر می‌شود. هر بار که این بلاک اجرا شود:

- مقادیر ورودی به پارامترها اختصاص داده می‌شوند
- بدنه‌ی تابع به صورت ترتیبی اجرا می‌شود

Scratch اجرای توابع را به صورت تفسیرشده (Interpreted) انجام می‌دهد.

۶.۱.۵ نگاشت مفهومی Scratch به C++

در فرآیند پیاده‌سازی Scratch با استفاده از زبان C++، مفاهیم بصری Scratch باید به سازه‌های متنی و الگوریتمی تبدیل شوند. این نگاشت به شکل زیر انجام می‌شود:

- Custom Block در Scratch معادل یک تابع (Function) یا متد (Method) در C++ است.
- Define Block که برای تعریف تابع استفاده می‌شود، معادل تعریف تابع (Function Definition) در C++ است.
- Parameters در Scratch معادل پارامترهای ورودی تابع (Function Arguments) در C++ هستند.
- Block Body یا بدنه‌ی بلاک معادل یک لیست از دستورات (List of Commands) در C++ است که به صورت ترتیبی اجرا می‌شوند.
- Call Block که باعث اجرای یک Custom Block می‌شود، معادل فراخوانی تابع (Function Call) در C++ است.

۷.۱.۵ مدل‌سازی بلاک‌ها در C++

برای پیاده‌سازی Scratch، هر بلاک می‌تواند به صورت یک ساختار داده مدل شود. نمونه‌ی ساده:

```
enum struct BlockType { Move, Turn, Repeat, If, CustomFunctionCall };

struct Block {
    BlockType type;
    std::vector<Block*> children;
    std::vector<Value> parameters;
};
```

۶ منوی Costumes

۱.۶ آپلود تصویر

۱.۱.۶ امکان آپلود تصویر (Sprite / Background)

در این بخش، کاربر می‌تواند تصویر دلخواه خود را برای استفاده به‌عنوان کاراکتر (Sprite) یا پس‌زمینه (Background) وارد برنامه کند.

مراحل انجام توسط کاربر

۱. کاربر از منوی مربوطه، گزینه Upload Image را انتخاب می‌کند.

۲. پنجره انتخاب فایل باز می‌شود و کاربر یک فایل تصویری (مانند PNG یا JPG) را انتخاب می‌کند.

۳. کاربر این امکان را دارد که چند عکس را با هم آپلود کند.

۴. پس از تأیید، تصویر وارد برنامه شده و در لیست Spriteها یا پس‌زمینه‌ها نمایش داده می‌شود.

قابلیت‌های پس از آپلود کاربر می‌تواند تصویر واردشده را انتخاب کند و:

- موقعیت آن را در صحنه مشخص کند.
- اندازه آن را تغییر دهد.

۲.۶ Animate کردن عکس آپلودشده

به بیان دیگر این بخش به نوعی حرکات را با نامی خاص ذخیره می‌کند تا در بخش کد با فراخوانی آنها بتوان کدنویسی راحت‌تری را تجربه کرد. در ورژن اصلی این قابلیت وجود ندارد و شما می‌توانید از خلاقیت خود برای پیاده‌سازی هرچه بهتر این بخش کمک بگیرید.
چیزی که مد نظر است:

- کاربر عکسی را انتخاب کند.
- با زدن دکمه‌ای حرکات عکس شروع به ضبط شدن بشوند.
- در این فاصله کاربر عکس را جابجا میکند.
- بعد از اتمام جابجایی دکمه‌ای را جهت اتمام ضبط می‌زند.
- این حرکت را برای عکس مدنظر با نام خاصی ذخیره کرده و در بخش کد نویسی هر زمان که با آن عکس کار می‌کند توانایی فراخوانی این حرکت ضبط شده را هم خواهد داشت.

۳.۶ ابزارهای طراحی و ویرایش تصویر (قلم، پاک‌کن و ...)

این بخش به کاربر اجازه می‌دهد بدون نیاز به نرم‌افزار خارجی، تصویر Sprite یا فریم‌های آن را ویرایش کند.

مراحل انجام توسط کاربر

۱. کاربر یک Sprite یا یک فریم را برای ویرایش انتخاب می‌کند.

۲. وارد بخش Edit / Paint Mode می‌شود.

۳. یکی از ابزارهای طراحی را انتخاب می‌کند.

۴. کاربر با ماوس روی بوم طراحی کرده و تغییرات را مشاهده می‌کند.

۵. پس از پایان، تغییرات ذخیره می‌شوند و تصویر ویرایش‌شده جایگزین نسخه قبلی می‌شود.

ابزارهای موجود

- **قلم (Pen):** برای کشیدن خطوط و طراحی دستی
  - امکان انتخاب اندازه و رنگ قلم (۳ اندازه و ۳ رنگ)
- **پاک‌کن (Eraser):** برای حذف بخش‌هایی از تصویر
- **سطل رنگ (Fill):** برای رنگ کردن Background
- **متن (Text):** امکان افزودن متن (text)
- **ابزار اشکال ساده:** مانند خط، دایره یا مستطیل
- **تنظیم موقعیت دقیق:** امکان مشخص کردن موقعیت یک Sprite با وارد کردن x,y
- **پاک کردن کل صفحه:** امکان حذف کامل محتوای بوم طراحی

۱.۳.۶ برگرداندن افقی و عمودی تصویر (Flip)

این قابلیت برای ساخت حرکات متقارن، به‌خصوص حرکت به چپ و راست کاراکترها، بسیار کاربردی است و نیاز به طراحی مجدد تصویر را از بین می‌برد.

مراحل انجام توسط کاربر

۱. کاربر یک Costume یا فریم را انتخاب می‌کند.

۲. گزینه Flip Horizontal یا Flip Vertical را انتخاب می‌کند.

۳. نتیجه تغییر جهت تصویر را مشاهده می‌کند.

۴. پس از پایان، تغییرات ذخیره می‌شوند و تصویر ویرایش‌شده جایگزین نسخه قبلی می‌شود.

۷ منوی تنظیمات

۱.۷ تنظیمات پس‌زمینه

در این بخش از پروژه، هدف ایجاد سیستمی است که کاربر بتواند پس‌زمینهٔ محیط بصری پروژهٔ خود را مدیریت کند. این بخش شامل سه قابلیت اصلی است:

* وجود تصاویر پیش‌فرض پس‌زمینه
* امکان انتخاب پس‌زمینه از سیستم
* امکان رسم/ویرایش پس‌زمینه با قلم و قابلیت‌های رسم نرم‌افزار

نکته مهم: پس از تغییر پس‌زمینه، نباید پس‌زمینه روی المان‌های از پیش قرار داده شده قرار بگیرد و باید در پایین‌ترین لایه رندر شود.

۱.۱.۷ منوی دسترسی به تنظیمات پس‌زمینه

برای شروع، باید یک دکمه یا آیکون در منوی اصلی (یا در محل دلخواه) تعبیه کنید. می‌توانید آیکون هر کدام از قابلیت‌ها را به صورت یک دکمه جداگانه در منوی اصلی برنامه قرار دهید.

* می‌توانید بقیه قابلیت‌ها را در قالب یک زیرمنو برای منوی اصلی تنظیمات پس‌زمینه ایجاد کنید.
* نام پس‌زمینهٔ فعلی را در منوی تنظیمات پس‌زمینه نمایش دهید.

۲.۱.۷ کتابخانه تصاویر پس‌زمینه پیش‌فرض

برنامه باید تعدادی تصویر آماده داشته باشد تا کاربر بتواند با انتخاب آن‌ها تصویر پس‌زمینه را تغییر دهد.

* با انتخاب این قابلیت، تصاویر زمینهٔ پیش‌فرض به کاربر نشان داده می‌شود و کاربر می‌تواند با انتخاب هر کدام، تصویر مد نظر را به عنوان پس‌زمینه قرار دهد.
* قرار دادن **۲ تصویر پس‌زمینهٔ پیش‌فرض** برای این بخش کافی است.
* هنگام نمایش پیش‌نمایش تصاویر، نام هر عکس نیز در کنار آن نشان داده شود.

۳.۱.۷ انتخاب تصویر پس‌زمینه از سیستم

این ویژگی به کاربر اجازه می‌دهد پس‌زمینه‌های مد نظر خود را از سیستم به برنامه وارد کند. با انتخاب این قابلیت باید پنجرهٔ مدیریت فایل ویندوز باز شود تا کاربر بتواند تصویر مورد نظر را انتخاب کند و آن را به عنوان تصویر زمینه قرار دهد.

* **(جبرانی)** پس از انتخاب هر تصویر توسط کاربر، آن تصویر به پوشهٔ تصاویر پیش‌فرض منتقل شود و از طریق منوی تصاویر پیش‌فرض، بتوان به تصاویری که قبلاً در برنامه استفاده شده‌اند دسترسی داشت.
* **توجه:** تصویر انتخابی باید به اندازهٔ پنجرهٔ اصلی برنامه **تغییر اندازه** داده شود.

۴.۱.۷ ویرایشگر تصویر داخلی

یکی از بخش‌های جالب برنامه، امکان ترسیم مستقیم پس‌زمینه است. کاربر با انتخاب این قابلیت باید بتواند وارد قسمت ادیت تصاویر شود و یک تصویر زمینه را با قلم بکشد. تمام قابلیت‌هایی که در بخش دستورات ترسیمی پیاده‌سازی کرده‌اید، باید در این بخش قابل دسترسی باشد. تصویر ایجاد شده توسط کاربر بعد از تأیید، باید به عنوان تصویر زمینه قرار بگیرد.

* **(جبرانی)** پس از تأیید هر تصویر، آن را در پوشهٔ تصاویر پیش‌فرض ذخیره کنید و امکان وارد کردن نام دلخواه برای تصویر را نیز در نظر بگیرید.
* **(جبرانی)** کاربر بتواند ترسیمات مدنظر خود را روی تصویر زمینهٔ فعلی اعمال کند و آن را ذخیره کند.

۲.۷ بخش مدیریت کاراکترها (Sprite Manager)

در محیط Scratch، تمام کاراکترهای پروژه در بخشی مجزا (Sprite Pane) نمایش داده می‌شوند و کاربر می‌تواند به‌سادگی بین آن‌ها جابه‌جا شود. در این پروژه نیز، بخش مدیریت کاراکترها باید نقش مشابهی ایفا کند و مسئول مدیریت تمامی کاراکترهای قابل‌نمایش در صحنهٔ اصلی باشد.

۱.۲.۷ نمایش آیکون‌محور کاراکترها

تمامی کاراکترهای تعریف‌شده در پروژه باید در یک پنل مجزا (مانند نوار کناری یا نوار پایینی محیط) به‌صورت تصویر کوچک یا آیکون نمایش داده شوند. این نمایش باید به کاربر امکان دهد در یک نگاه، تعداد و وضعیت کلی کاراکترهای پروژه را مشاهده کند؛ مشابه پنل Sprites در Scratch.

۲.۲.۷ انتخاب کاراکتر فعال

با کلیک روی آیکون هر کاراکتر، آن کاراکتر به‌عنوان کاراکتر فعال انتخاب می‌شود. کاراکتر فعال، کاراکتری است که تنظیمات آن (مانند موقعیت، اندازه یا نام) در بخش تنظیمات نمایش داده شده و قابل ویرایش است. در Scratch نیز تمام تغییرات همواره روی Sprite انتخاب‌شده اعمال می‌شوند.

۳.۲.۷ نام‌گذاری و شناسایی کاراکتر

هر کاراکتر باید دارای یک نام مشخص و منحصربه‌فرد باشد. کاربر باید بتواند نام کاراکتر را از طریق رابط کاربری ویرایش کند. این نام‌گذاری به کاربر کمک می‌کند، به‌ویژه در پروژه‌های بزرگ‌تر، کاراکترها را به‌راحتی از یکدیگر تشخیص دهد؛ همان‌طور که در Scratch هر Sprite دارای نام قابل تغییر است.

۴.۲.۷ تنظیم موقعیت کاراکتر در صحنه

موقعیت هر کاراکتر باید بر اساس مختصات افقی (x) و عمودی (y) قابل تنظیم باشد. تغییر موقعیت می‌تواند از دو طریق انجام شود:

* وارد کردن مستقیم مقادیر عددی مختصات
* جابه‌جایی مستقیم کاراکتر با ماوس در صحنهٔ اصلی

این رفتار مشابه Scratch است که کاربر می‌تواند Sprite را هم با کشیدن و هم با تغییر مقادیر مختصات جابه‌جا کند.

۵.۲.۷ تغییر اندازه و جهت (چرخش)

کاربر بتواند اندازهٔ کاراکتر را تغییر دهد (بزرگ یا کوچک کند). همچنین امکان تنظیم زاویهٔ چرخش کاراکتر نیز فراهم باشد. این تنظیمات باید تأثیر بصری فوری در صحنه داشته باشند تا کاربر نتیجهٔ تغییرات را بلافاصله مشاهده کند، همانند تنظیمات Size و Direction در Scratch.

۶.۲.۷ نمایش یا مخفی‌سازی کاراکتر

برای هر کاراکتر، گزینه‌ای جهت مخفی یا نمایان‌کردن آن در صحنه وجود داشته باشد. مخفی‌سازی تنها باید باعث عدم نمایش کاراکتر شود و نباید آن را از پروژه حذف کند. این قابلیت مشابه بلوک‌های hide و show در Scratch است، با این تفاوت که از طریق رابط کاربری گرافیکی کنترل می‌شود.

۷.۲.۷ حذف کاراکتر

کاربر بتواند یک کاراکتر را به‌صورت دائمی از پروژه حذف کند. پیش از حذف، سیستم باید با نمایش پیام هشدار، تأیید کاربر را دریافت کند. این رفتار مشابه حذف Sprite در Scratch است که با آیکون سطل زباله انجام می‌شود.

۸.۲.۷ افزودن کاراکتر جدید

دکمه‌ای مشخص برای افزودن کاراکتر جدید در رابط کاربری وجود داشته باشد. با انتخاب این گزینه، کاربر یکی از حالت‌های زیر را انتخاب می‌کند:

- انتخاب کاراکتر از کتابخانهٔ داخلی (مشابه Scratch Sprite Library)
- ساخت کاراکتر سفارشی (مانند بارگذاری تصویر یا طراحی ساده)
- انتخاب تصادفی یک کاراکتر آماده (Random Sprite)

۸ منوی اجرای کد

۱.۸ اجرای کد در Scratch و مدل اجرای آن در C++ با SDL

یکی از تفاوت‌های اساسی Scratch با زبان‌های برنامه‌نویسی متنی، نحوه‌ی اجرای کد است. در Scratch، برنامه‌ها به صورت بصری و مبتنی بر بلاک‌ها ساخته می‌شوند و اجرای آن‌ها توسط یک موتور تفسیرکننده انجام می‌گیرد. در این بخش، ابتدا نحوه‌ی اجرای کد در Scratch بررسی می‌شود و سپس یک مدل اجرایی مناسب برای پیاده‌سازی Scratch با زبان C++ و کتابخانه SDL ارائه می‌گردد.

۱.۱.۸ نحوه اجرای کد در Scratch

Scratch یک زبان تفسیری (Interpreted) و رویدادمحور (Event-Driven) است. در این محیط، کد به صورت خطی نوشته نمی‌شود، بلکه از طریق اتصال بلاک‌ها یک ساختار اجرایی شکل می‌گیرد.

شروع اجرای برنامه در Scratch، برنامه معمولاً از طریق یکی از بلاک‌های رویدادی آغاز می‌شود، مانند:

- when green flag clicked
- when key pressed
- when sprite clicked

این بلاک‌ها نقش نقطه‌ی ورود (Entry Point) برنامه را دارند و بدون آن‌ها اجرای کد آغاز نمی‌شود.

مدل اجرایی Scratch مدل اجرای Scratch شامل ویژگی‌های زیر است:

- **اجرای ترتیبی بلاک‌ها:** بلاک‌ها از بالا به پایین و به صورت ترتیبی اجرا می‌شوند. هر بلاک پس از اتمام اجرای بلاک قبلی اجرا می‌شود، مگر اینکه بلاک‌های کنترلی مانند حلقه یا شرط وجود داشته باشند.
- **اجرای هم‌زمان (Concurrency):** Scratch امکان اجرای هم‌زمان چند اسکریپت را فراهم می‌کند. برای مثال، چند بلاک رویدادی مختلف می‌توانند به طور هم‌زمان فعال شوند. Scratch این هم‌زمانی را به صورت سبک (Lightweight Threads) مدیریت می‌کند.
- **اجرای تفسیرشده:** Scratch کد را به صورت مستقیم (کامپایل‌شده) اجرا نمی‌کند، بلکه:
  - هر بلاک به عنوان یک دستور تفسیر می‌شود.
  - موتور Scratch تصمیم می‌گیرد هر بلاک چه عملی انجام دهد.

  در نتیجه، Scratch به کامپایل کامل برنامه نیازی ندارد.

مدیریت حالت (State) در Scratch حالت برنامه شامل موارد زیر است:

- موقعیت و وضعیت Spriteها
- مقادیر متغیرها
- وضعیت صدا و ظاهر

تمام این اطلاعات در طول اجرای برنامه به صورت پویا تغییر می‌کنند.

نقش موتور اجرا (Execution Engine) Scratch دارای یک موتور اجرا است که وظایف زیر را بر عهده دارد:

- مدیریت اسکریپت‌های فعال
- اجرای بلاک‌ها به ترتیب
- مدیریت هم‌زمانی
- کنترل توقف و ادامه اجرای برنامه

این موتور به صورت یک Loop مرکزی عمل می‌کند.

۲.۱.۸ مدل پیشنهادی اجرای کد در C++ برای پیاده‌سازی Scratch

در C++ لازم است یک موتور اجرای مشابه طراحی شود که بتواند بلاک‌ها را تفسیر کند. توصیه می‌شود این بخش در یک ماژول/تابع جدا (مثلاً با نام Engine) پیاده‌سازی شود تا بتوان در بخش‌های مختلف برنامه از آن استفاده کرد.

ساختار کلی موتور اجرا موتور اجرا در برنامه C++ می‌تواند شامل اجزای زیر باشد:

- Event Manager
- Script Scheduler
- Block Interpreter
- Global State Manager

نقطه شروع اجرای برنامه در C++ برخلاف Scratch که اجرای برنامه با بلاک آغاز می‌شود، اجرای برنامه از تابع main شروع می‌شود. در این پروژه:

- تابع main حلقه‌ی اصلی SDL را اجرا می‌کند.
- رویدادهای ورودی کاربر بررسی می‌شوند.
- در صورت فعال شدن یک رویداد، اسکریپت مرتبط اجرا می‌شود.

اجرای بلاک‌ها در C++ (Interpreter) در مدل تفسیرکننده، هر بلاک به عنوان یک دستور تفسیر می‌شود. ساختار کلی اجرای یک بلاک می‌تواند به صورت زیر باشد:

```
void executeBlock(Block* block, Context& context) {
    switch (block->type) {
        case BlockType::Move:
            executeMove(block, context);
            break;
        case BlockType::If:
            executeIf(block, context);
            break;
        case BlockType::Repeat:
            executeRepeat(block, context);
            break;
```

```
        case BlockType::CustomFunctionCall:
            executeFunctionCall(block, context);
            break;
    }
}
```

۲.۸ توقف موقت اجرای کد (Pause / Resume)

در این بخش باید یک دکمه در منوی اجرای کد برای **توقف موقت** اجرای کد ایجاد کنید. منظور از توقف موقت، توقف برنامه اصلی یا از دست رفتن پیشرفت در اجرای برنامه در حال اجرا نیست. کاربر با این قابلیت باید بتواند کد را در هر مرحله‌ای که قرار دارد متوقف کند و منتظر دستور ادامه (دکمه اجرا) بماند تا اجرای کد از محل توقف ادامه پیدا کند.

۱.۲.۸ نکات اجرایی

- ایستگاه‌های توقف را **بعد از هر بلوک** در نظر بگیرید (نیازی به توقف کد در میان اجرای فرمان‌های داخل یک بلوک نیست).

۲.۲.۸ پیشنهاد پیاده‌سازی

برای پیاده‌سازی این ویژگی می‌توانید بلوک‌ها را اندیس‌گذاری کنید و به تابع اجرای آن‌ها یک ورودی برای «اندیس اولین بلوکی که اجرا خواهد شد» اضافه کنید. در هنگام اجرا نیز اندیس قسمت‌های اجراشده را ذخیره کنید. هنگام دریافت دستور توقف، با ایجاد یک شرط، برنامه را متوقف کنید و پس از دستور اجرا، از اندیس اولین بلوک اجرا نشده برنامه را ادامه دهید.

۳.۲.۸ الزامات رابط کاربری

- هنگام توقف موقت برنامه، **تمام منوها و بخش‌های نرم‌افزار باید قابل دسترسی باشند**.
- هنگام توقف موقت برنامه، **نباید پنجره پیش‌نمایش برنامه به حالت اولیه برگردد** یا تغییر ناخواسته‌ای داشته باشد.

۴.۲.۸ موارد جبرانی

- آیکون اجرای برنامه را پس از کلیک به آیکون Pause تغییر دهید و هنگام توقف موقت برنامه مجدداً آن را به حالت اولیه برگردانید.
- دستورات اجرا شده و نشده را در حین اجرای برنامه با روش دلخواه (کم‌رنگ شدن، ایجاد یک نشانگر روی آخرین بلوک اجرا شده و ...) از یکدیگر متمایز کنید.

۹ بخش مدیریت صدا (Sound Manager)

در Scratch، صداها یکی از عناصر کلیدی پروژه هستند و در بخشی جداگانه مدیریت می‌شوند. در این پروژه نیز، بخش مدیریت صدا باید امکان افزودن و کنترل صداها را به‌صورت بصری فراهم کند.

۱.۹ افزودن صدا

- دکمه‌ای با عنوان «افزودن صدا» در رابط کاربری وجود داشته باشد.
- با انتخاب این دکمه، گزینه‌های زیر نمایش داده شوند:
  - انتخاب صدا از میان صداهای آماده در کتابخانهٔ داخلی (مشابه Scratch Sound Library)
  - انتخاب تصادفی یک صدا از میان صداهای موجود

۲.۹ کنترل ویژگی‌های صدا

برای هر صدای اضافه‌شده، امکانات زیر در نظر گرفته شود:

- mute / unmute: قطع یا وصل موقت صدا بدون حذف آن از پروژه
- **تنظیم بلندی صدا:** تغییر حجم صدا در یک بازهٔ مشخص (مثلاً 0% تا 100%)

این تنظیمات باید به‌صورت تعاملی و از طریق عناصر گرافیکی (مثل اسلایدر یا دکمه) انجام شوند.
توجه: همانند بخش Sprite، این قسمت صرفاً مسئول مدیریت و تنظیم صدا از دید کاربر است. اجرای واقعی صدا در زمان اجرای پروژه خارج از محدودهٔ این مرحله محسوب می‌شود. همچنین Scratch در بخش مدیریت صدا تنظیمات بیشتری را پوشش می‌دهد که نیازی به پیاده‌سازی آن‌ها نیست.