گزارش تمرین ۱ شبیه سازی در فیزیک

علی ستاره کوکب
شماره دانشجویی: ۹۵۱۰۰۴۹۱
دانشگاه صنعتی شریف
دانشکده فیزیک

۷ مهر ۱۳۹۹

۱ مقدمه

در این تمرین به بررسی اتوماتای سلولی[۱]می پردازیم. در سه تمرین اول اتوماتای سلولی یک بعدی را بررسی می کنیم و در تمرین آخر به بررسی بازی زندگی[۲] خواهیم پرداخت. چهار جز اصلی هر اتوماتای سلولی شبکه ی سلول ها، حالت سلول ها، همسایگی و قانونی است که به وسیله ی آن وضعیت بعدی سلول ها مشخص می شود.

۲ قانون کلاه

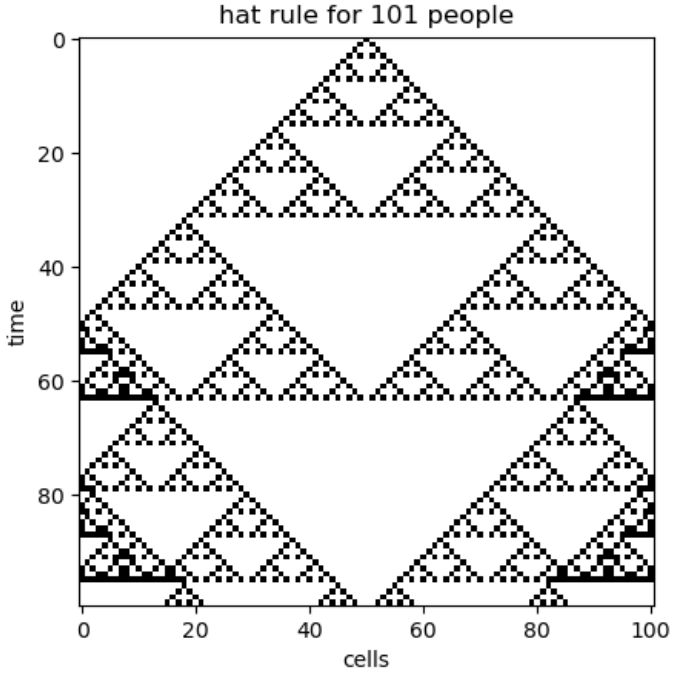
در این تمرین یک صف متشکل از ۲۰۱ نفر داریم که نفر وسط صف کلاه بر سر دارد و سایر افراد در ابتدا بدون کلاه می باشند. در این جا همسایه ی هر سلول، سلول سمت راست و چپ آن می باشد و قانونی که تعیین می کند در زمان بعد کدام نفرات کلاه بر سر بگذارند بدین شکل است که اگر تنها یکی از همسایه های هر سلول کلاه داشته باشد، در زمان بعدی آن سلول نیز کلاه بر سر خواهد گذاشت و اگر هر دو همسایه ها کلاه داشته باشند یا هر دو بدون کلاه باشند در زمان بعد، سلول مورد بررسی کلاه بر سر نخواهد گذاشت. با دانستن مساله به بررسی کد ها می پردازیم.

در ابتدا تابعی تحت عنوان rule تعریف می کنیم. این تابع دو ورودی دارد که بیانگر حالت همسایگی های سلول مورد بررسی می باشد و به عنوان خروجی، بسته به حالت همسایه ها، حالت بعدی سلول مورد بررسی را برمی گرداند. پس از تعریف این تابع به بدنه اصلی برنامه

---

[۱] cellular automaton
[۲] game of life

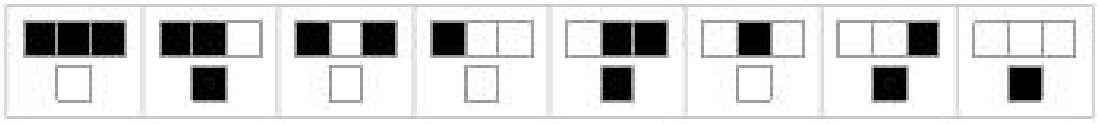
می رسیم. متغیر cell، متغیری است که حالت سلول ها را در آن نگه می داریم. cell temp متغیری است که محتویات cell را در آن کپی می کنیم. این بخاطر آن است که ما می خواهیم همه ی سلول ها را باهم همزمان آپدیت کنیم بنابراین نیاز است که یک نسخه از حالت سلول ها در مرحله قبل داشته باشیم. در حلقه ی اولیه for حالت اولیه سیستم را مشخص می کنیم. متغیر matrix cell ماتریسی است که در آن تاریخچه ی حالت همه ی سلول ها را در آن نگه داری می کنیم. برای اجرای درست شبیه سازی، ما نیاز به دو حلقه داریم. یک حلقه برای گام ها زمانی، و یک حلقه ی دوم داخل آن برای آن که حالت هر سلول را آپدیت کنیم. با توجه به آنکه شرایط مرزی خواسته شده، شرایط مرزی تناوبی است، گفته شده که اگر به سلول آخر رسیدیم، همسایه بعدی را خانه ی صفر در نظر بگیریم. در هر مرحله نیز برای هر سلول، تابع rule را صدا می زنیم تا حالت سلول را در زمان بعدی مشخص کنیم. پس اتمام این دو حلقه متغیر G تعریف شده است. این متغیر یک ماتریس می باشد که در آن رنگ متناسب با هر سلول وجود دارد. در اینجا تعیین می کنیم که اگر حالت سلول برابر صفر بود، رنگ سلول را برابر سفید، و اگر حالت سلول برابر یک بود، رنگ سلول را برابر سیاه قرار بده. در نهایت نیز با استفاده از متد های موجود در کتابخانه ی matplotlib شکل مورد نظر را رسم می کنیم که در شکل ۱ آن را می بینید .



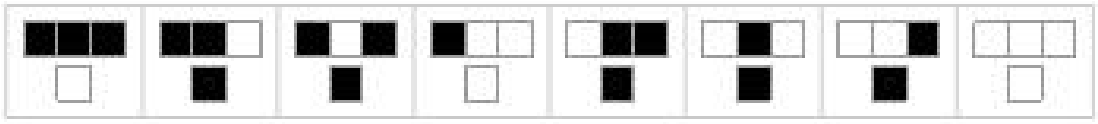
شکل ۱: شکل نهایی قانون کلاه برای ۱۰۱ سلول و ۱۰۰ گام زمانی سلول های روشن سیاه و سلول ها خاموش سفید می باشند

۳ قوانین ۲۵۶ گانه

در این تمرین دو قانون ۷۵ و ۱۱۰ را از میان قوانین ۲۵۶ گانه شبیه سازی می کنیم. روند کلی کد های این تمرین نیز مانند تمرین اول می باشد. ابتدا تابع rule را تعریف می کنیم که در آن قوانین مربوط به سلول ها در آن تعیین می شود. در اینجا برخلاف مساله ی قبل، این تابع سه ورودی دارد چرا که در این مسائل، حالت سلولی که در آن حضور داریم نیز در تعیین حالت بعدی آن موثر است بنابراین ورودی های این تابع همسایه ی سمت راست، سلول کنونی و همسایه سمت چپ می باشد. شکل شماتیک قانون ۱۱۰ و قانون ۷۵ را در شکل زیر می بینید. این دوشکل شماتیک به وسیله دستور rule plot متمتیکا بدست آمده اند. در شکل های زیر سلول های روشن سیاه و سلول ها خاموش سفید می باشند.



شکل ۲: شکل شماتیک قانون ۷۵



شکل ۳: شکل شماتیک قانون ۱۱۰

بدنه ی اصلی کد در این تمرین نیز مانند تمرین اول می باشد. یک حلقه ی تو در تو داریم که حلقه ی اول مربوط به گام زمانی و حلقه ی دوم، مربوط به بررسی حالت سلول ها در آن گام زمانی می باشد. شکل نهایی این دو قانون را برای ۲۰۱ سلول و ۳۰۰ گام زمانی را می توانید در شکل ۴ و ۵ ببینید.

۴ Wolfram Classification

در این تمرین با تغییر یک صفر به یک یا برعکس در فرم دودویی قانون ۱۱۰، می خواهیم ببینیم چه شکل هایی ایجاد می شوند. از آنجا که فرم دودویی این قانون از ۸ رقم تشکیل شده است

بنابراین ۸ قانون مختلف برای این قسمت خواهیم داشت. کلیت کد این تمرین نیز مانند دو تمرین قبل می باشد فقط اندکی تغییرات برای راحت تر شدن کار اعمال شده اند. در اینجا برای آنکه مجبور نباشیم هشت بار جداگانه کد را بنویسیم، یک ورودی به تابع rule که در بردارنده ی قانون مورد نظر می باشد، اضافه کرده ایم. این ورودی چهارم، یک لیست می باشد که حاوی فرم دودویی قانون مورد نظر می باشد. بدین شکل در این برنامه کافی است فرم دودویی قانون مورد نظر به برنامه بدهیم تا قانون مورد نظرمان اعمال شود. در بدنه ی اصلی کد نیز یک حلقه دیگر اضافه شده است. در این حلقه هر بار یکی از هشت قانون مورد بررسی یه عنوان قانون حاکم بر سلول ها در نظر گرفته می شود. در نهایت نیز با استفاده از متد های موجود در کتابخانه matplot . شکل ها را رسم می کنیم. (شکل های ۴ تا ۱۳) در تمام این شکل ها سلول های روشن سیاه و سلول ها خاموش سفید می باشند.

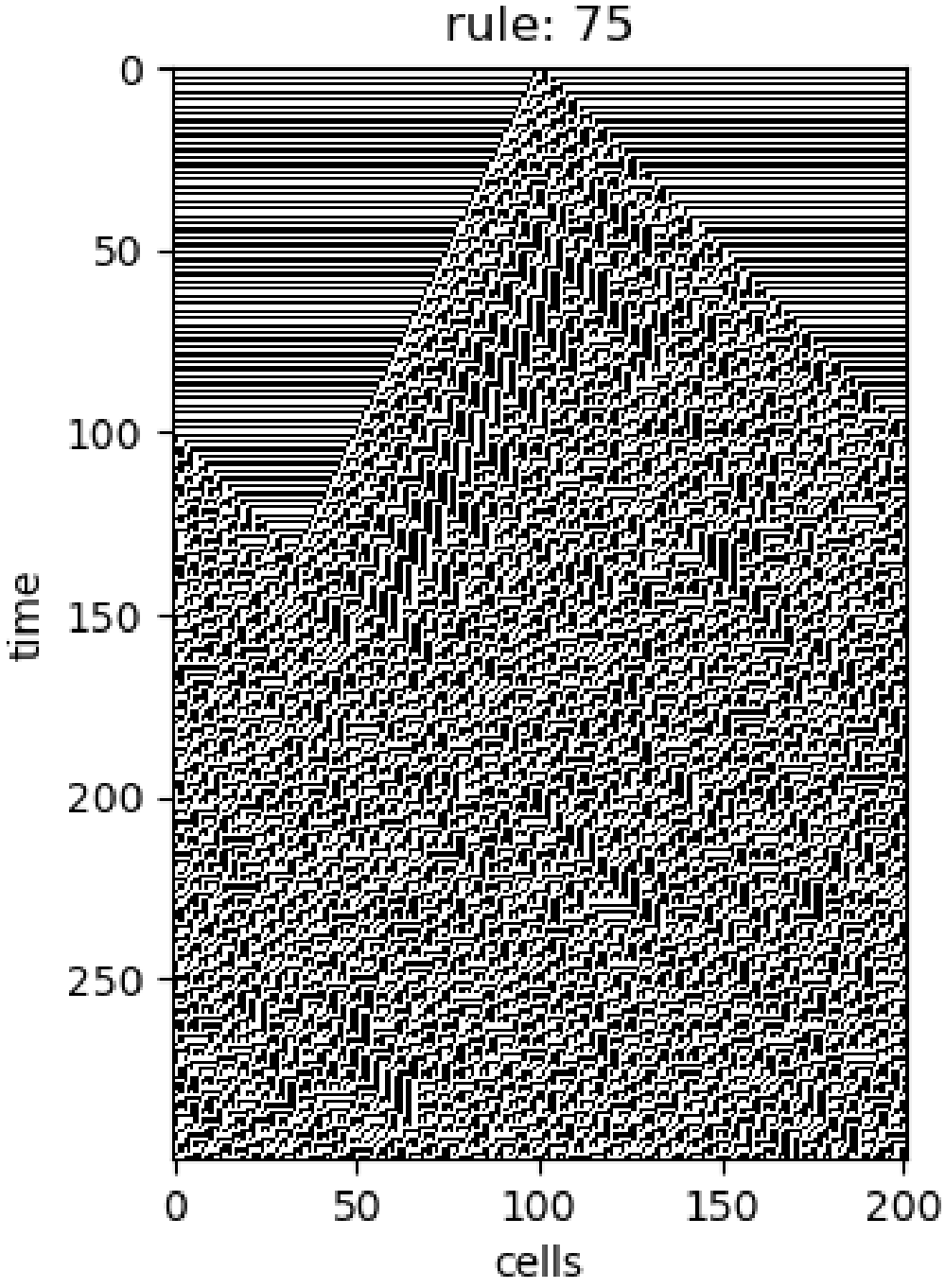
در قسمت بعدی سوال خواسته شده که هر کدام از این هشت شکل و دو شکل تمرین دوم را مطابق طبقه بندی ولفرم، طبقه بندی کنیم. در طبقه بندی ولفرم چهار کلاس وجود دارد: یکسانی، نوسانی، تصادفی و پیچیده. کلاس یکسانی، حالتی است که در آن تمام سلول ها به حالت صفر یا سفید می رسند. کلاس نوسانی، حالتی است که در آن یک الگو تکرار می شود. کلاس تصادفی، حالتی است که هیچ الگوی قابل مشاهده ای در سلول ها وجود ندارد. حالت پیچیده، حالتی است که بنظر اشکال ایجاد شده یک نظمی را دارا می باشند. با توجه به این توضیحات طبقه بندی این ده شکل، بصورت زیر می باشد:
حالت تناوبی: قانون ۴۶، قانون ۷۸، قانون ۱۰۶، قانون ۱۰۸، قانون ۱۱۱، قانون ۲۳۸
حالت پیچیده: قانون ۱۱۰، قانون ۱۲۶، قانون ۱۰۲
حالت تصادفی: قانون ۷۵

۵ Game of life

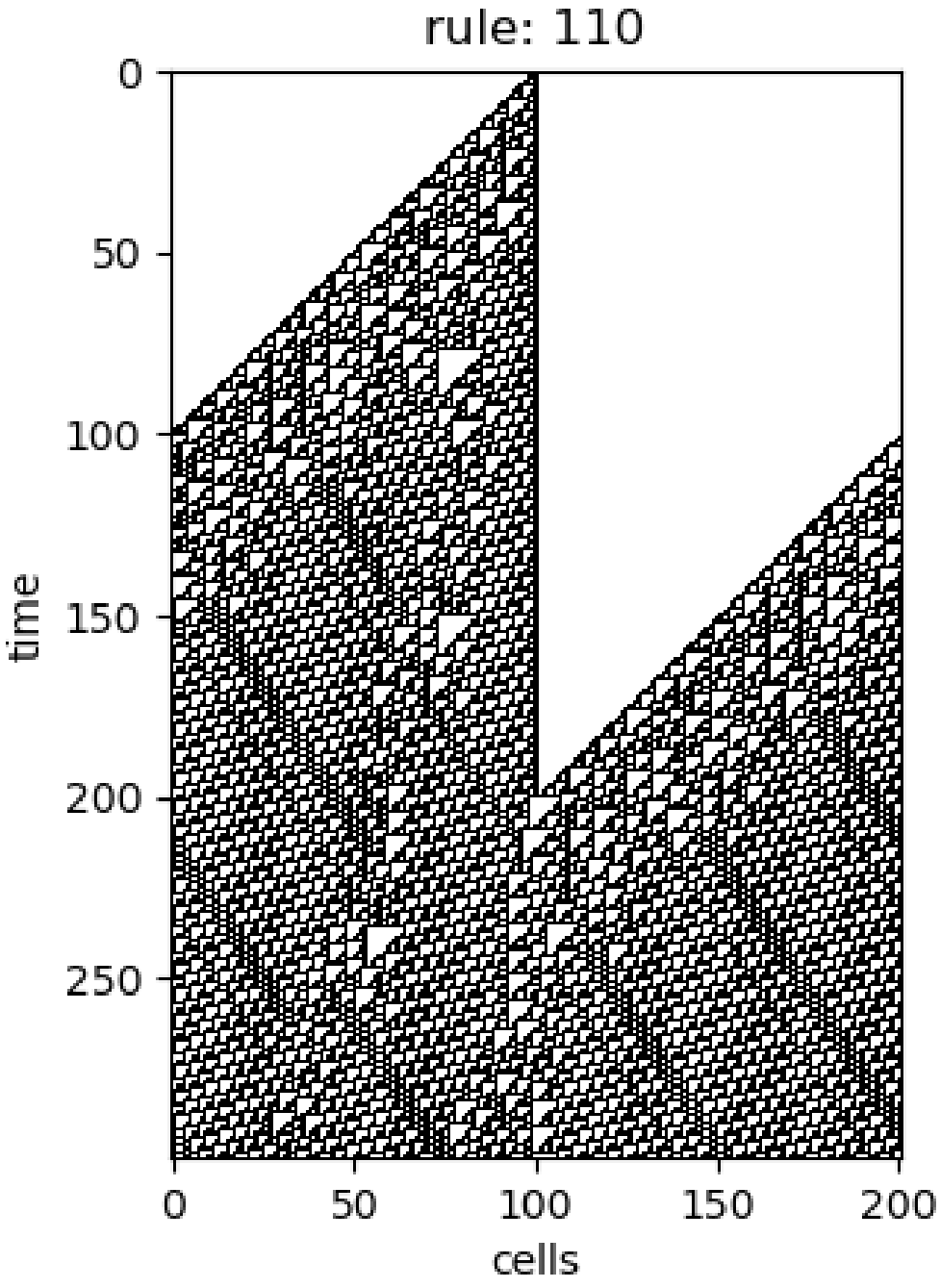
در این قسمت می خواهیم بازی زندگی را شبیه سازی کنیم. بازی زندگی یکی از معروف ترین مثال های اتوماتای سلولی دوبعدی می باشد. مانند هر مساله ی اتوماتای سلولی، پس از تعیین شبکه ی سلول ها (که در این جا یک شبکه ی دوبعدی است)، باید همسایگی را تعریف کنیم. در بازی زندگی همسایه هر سلول، نزدیک ترین هشت سلول به آن سلول در نظر گرفته می شوند. به عبارت دیگر، سلول های موجود در بالا، پایین، چپ، راست و همچنین چهار سلولی که بصورت قطری درمجاورت آن سلول وجود دارند، به عنوان همسایه در نظر گرفته می شوند. قوانین حاکم بر این مساله بدین صورت است: اگر سلولی روشن باشد و در همسایگی آن چهار یا بیش از چهار سلول و یا یک یا کمتر از یک سلول روشن باشند، سلول حاضر خاموش می شود. اگر سلولی خاموش باشد و دقیقا سه سلول همسایه آن روشن باشند، این سلول نیز روشن می شود؛ و در نهایت اگر هیچ یک از این حالات اتفاق نیفتد، سلول مورد نظر حالت فعلی خود را حفظ می کند. با دانستن مساله به بررسی کد آن می پردازیم.
همانند تمرین های پیش، در اینجا نیز تابع rule نقش اعمال کردن قوانین حاکم بر سلول ها را

دارد. این تابع دو ورودی دارد: ورودی اول یک آرایه ی دو بعدی است که وضعیت سلول ها در آن ذخیره می شود و ورودی دوم بیانگر طول یک ضلع شبکه می باشد (فرض می کنیم شبکه مربعی است). در داخل این تابع چهار حلقه ی تو در تو داریم. دو حلقه ی اول برای پیمایش کل خانه های شبکه می باشد. در اینجا چون شبکه دوبعدی است، بنابراین نیاز است که یک حلقه برای پیمایش در جهت سطر ها داشته باشیم و یک حلقه برای حرکت در جهت ستون ها. داخل این دوحلقه، دو حلقه ی دیگر برای پیمایش همسایه های هر سلول می باشد. بنابراین با عبور از دو حلقه ی اول گویی بر روی یکی از خانه های این شبکه قرار می گیریم و در دو حلقه ی بعدی هشت همسایه ی مجاور را در صورت روشن بودن، با متغیر neighbor می شماریم و در نهایت با بررسی شرط ها، وضعیت هر سلول را در زمان بعدی تعیین می کنیم. در نهایت نیز آرایه ی دوبعدی بروزرسانی شده را به عنوان خروجی تابع برمی گردانیم.

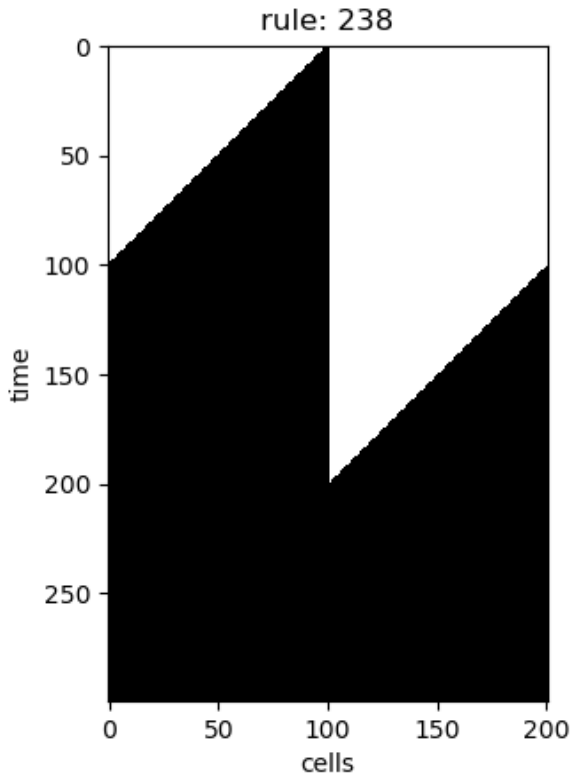
در سطور بعدی کد که بصورت کامنت درآمده اند، شرایط اولیه های خواسته شده در صورت مساله و همچنین یک شرط اولیه تصادفی آورده شده است. در صورت سوال خواسته شده که بازی زندگی را برای چهار حالت اولیه خاص بررسی کنیم که برای آنکه هر کدام از این چهار حالت را ببینیم، کافی حالت مورد نظر را از کامنت خارج کنیم تا آن شرط اولیه اعمال شود. دقت کنید که در هر زمان تنها یکی از حالت ها بصورت کد باشد و بقیه در حالت کامنت باشند. شرایط اولیه توسط یک حلقه در ماتریس مورد نظر جایگذاری می شود. پس از تعیین شرایط اولیه تعداد گام های زمانی و همچنین ابعاد شبکه تعیین می شوند. در حلقه ی بعدی، برای تعداد گام های زمانی مورد نظر بازی را اجرا می کنیم و در هر بار اجرا، کل حالت سیستم را در متغیر arr ذخیره می کنیم. به عبارت دیگر، این متغیر حاوی تمام تاریخچه ی سلول ها می باشد. برای آنکه بتوانیم انیمیشن تحول این سیستم را به تصویر بکشیم، از FuncAnimation در کتابخانه ی matlibpolt استفاده می کنیم. برای آنکه FuncAnimation به درستی کار کند نیاز است حداقل سه ورودی به آن بدهیم. ورودی اول شکلی است که این تابع وظیفه دارد آن را بروز رسانی کند که در اینجا آن را fig نام گذاشته ایم. متغیر دوم تابعی که عملا کار اصلی انیمیشن سازی را انجام می دهد. این تابع وظیفه دارد که به تعداد ورودی های آن (که عملا تعداد فریم های انیمیشن های ما می باشد) اجرا شود و در آخر متغیر interwal می باشد که بیانگر وقفه ای است که میان هر فریم وجود دارد. متغیر آخر blit که در اینجا آن را برابر True قرار داده ایم، برای آن است که صرفا آن نقاطی از شکل را در هر مرحله بکشیم که تغییر کرده اند و نیازی نباشد که در هر مرحله کل تصویر را بازسازی کنیم. تابعی که وظیفه ی انیمیشن سازی را دارد تابع updatefig می باشد. در این تابع در هرمرحله یک حالت سیستم را که در متغیر arr ذخیره شده را می خوانیم و نمایش می دهیم. متغیر i برای شمردن گام های زمانی است که پیش رفته ایم. دقت کنید که اگر می خواهید تعداد گام های زمانی را تغییر دهید، علاوه یر متغیر time بازه ی مجاز i را نیز تغییر دهید. در فیلم های ارسال شده، سلول های روشن با رنگ زرد نشان داده شده اند.



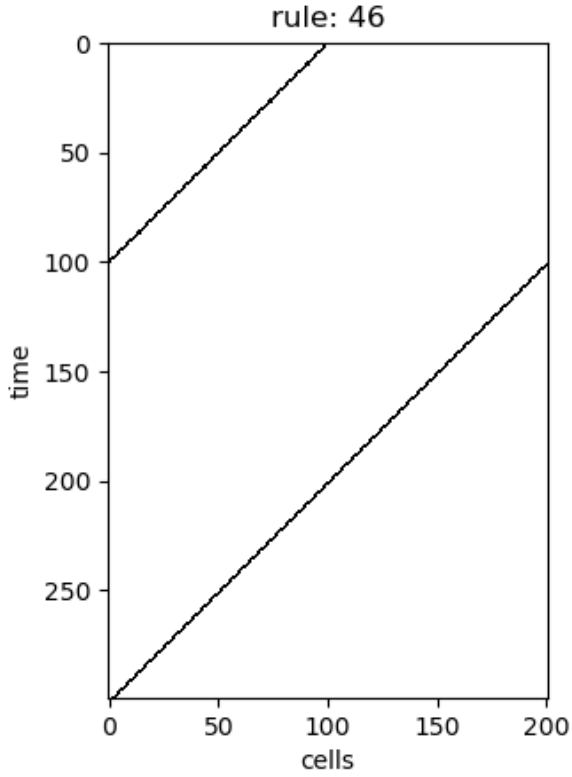
شکل ۴: شکل نهایی قانون ۷۵ برای ۲۰۱ سلول و ۳۰۰ گام زمانی



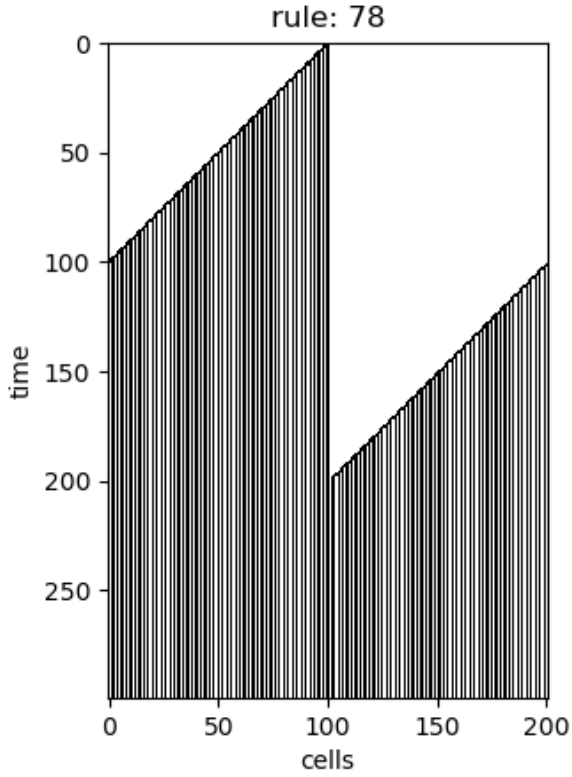
شکل ۵: شکل نهایی قانون ۱۱۰ برای ۲۰۱ سلول و ۳۰۰ گام زمانی



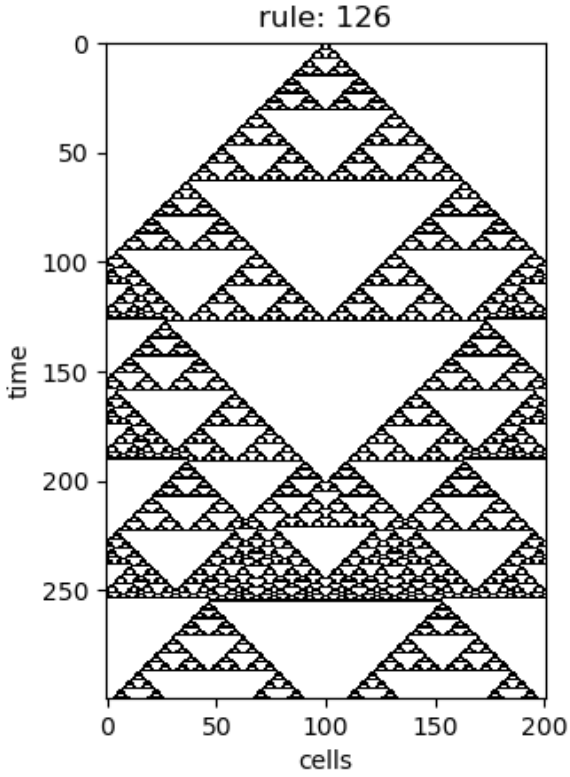
شکل ۶: شکل نهایی قانون ۲۳۸ برای ۲۰۱ سلول و ۳۰۰ گام زمانی



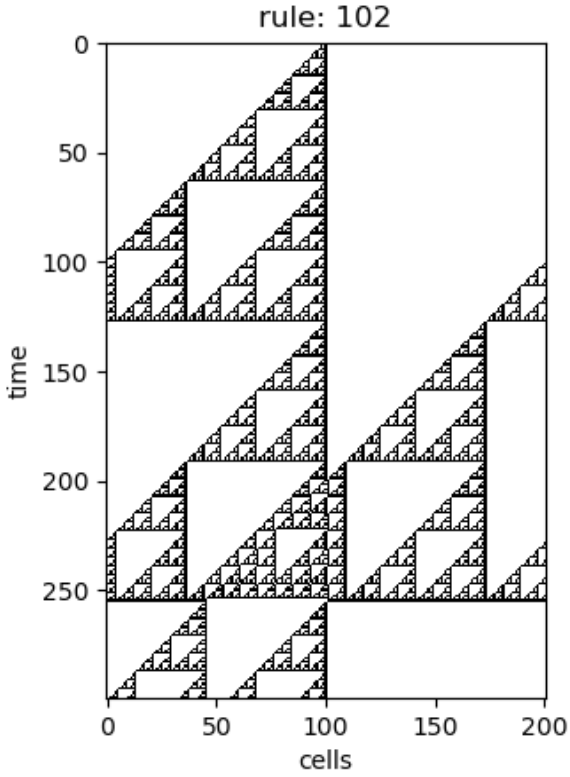
شکل ۷: شکل نهایی قانون ۴۶ برای ۲۰۱ سلول و ۳۰۰ گام زمانی



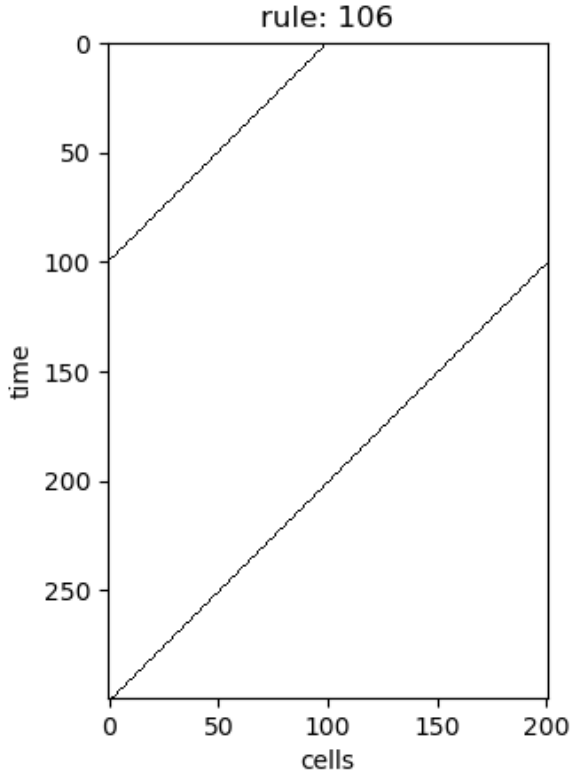
شکل ۸: شکل نهایی قانون ۷۸ برای ۲۰۱ سلول و ۳۰۰ گام زمانی



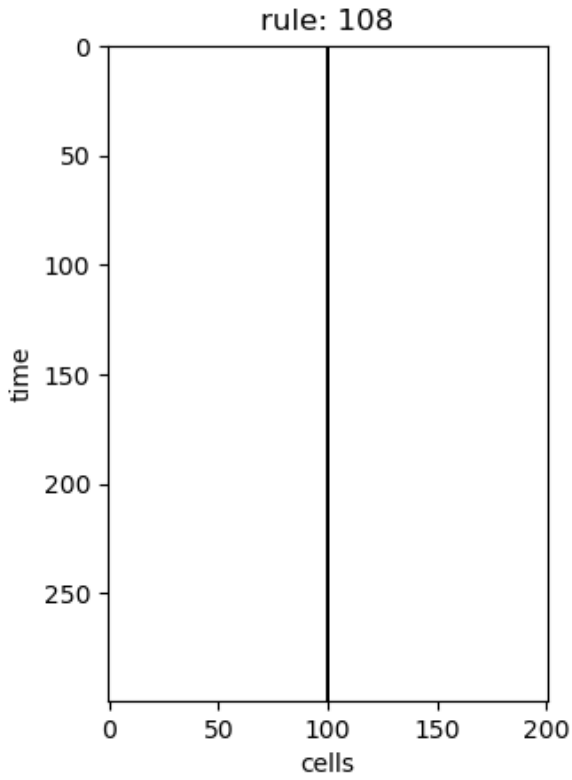
شکل ۹: شکل نهایی قانون ۱۲۶ برای ۲۰۱ سلول و ۳۰۰ گام زمانی



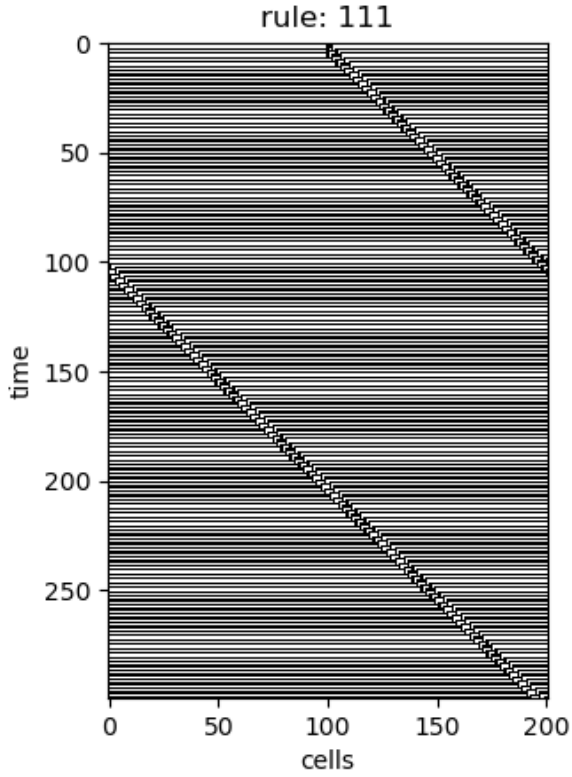
شکل ۱۰: شکل نهایی قانون ۱۰۲ برای ۲۰۱ سلول و ۳۰۰ گام زمانی



شکل ۱۱: شکل نهایی قانون ۱۰۶ برای ۲۰۱ سلول و ۳۰۰ گام زمانی



شکل ۱۲: شکل نهایی قانون ۱۰۸ برای ۲۰۱ سلول و ۳۰۰ گام زمانی



شکل ۱۳: شکل نهایی قانون ۱۱۱ برای ۲۰۱ سلول و ۳۰۰ گام زمانی